FORGEROCK®

Authentication and Single Sign-On Guide

/ ForgeRock Access Management 6.5

Latest update: 6.5.5

ForgeRock AS.
201 Mission St., Suite 2900
San Francisco, CA 94105, USA
+1 415-599-1100 (US)
www.forgerock.com

Abstract

Guide to working with authentication and single sign-on support. ForgeRock® Access Management provides authentication, authorization, entitlement and federation software.

Table of Contents

Preface

This guide covers concepts, implementation procedures, and customization techniques for working with the authentication and single sign-on features of ForgeRock Access Management.

This guide is written for anyone using AM to manage authentication, sessions, and implement single sign-on.

About ForgeRock Identity Platform™ Software

ForgeRock Identity Platform™ serves as the basis for our simple and comprehensive Identity and Access Management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of their employees and partners. For more information about ForgeRock and about the platform, see https://www.forgerock.com.

Chapter 1

Introducing Authentication and Single Sign-On

Authentication is the process of verifying login credentials submitted by a user or an entity comparing them to a database of authorized users. This guide covers how to set up, customize, and use the authentication process.

About Access Management and Authentication

Access management is about controlling access to resources using two processes: authentication and authorization.

Authentication is how AM verifies the identity of a user or an entity. *Authorization* is how AM determines whether a user has sufficient privileges to access a protected resource, and if so, access is granted to that user or entity. AM's authorization process is covered in the Authorization Guide.

AM plays a role similar to border control at an international airport. Instead of having each and every airline company deal with access to each destination, all airlines redirect passengers to border control. Border control then determines, or authenticates, the identity of each passenger according to passport credentials.

Redirect control also checks whether the identified passenger is authorized to fly to the destination corresponding to the ticket, perhaps based on visa credentials. Then, at the departure gate, an agent enforces the authorization from border control, allowing the passenger to board the plane as long as the passenger has not gotten lost, or tried to board the wrong plane, or swapped tickets with someone else. Thus, border control handles access management at the airport.

AM uses defined mechanisms to validate credentials and complete the authentication process. The authentication methods can vary. For example, AM is most frequently used to protect web application pages.

Consider a user who wants access to a protected web page. You can deploy an agent on the web application server. The agent redirects the user's request to an AM login page, where the user enters their credentials, such as username and password. AM determines who the user is, and whether the user has the right to access the protected page. AM then redirects the user back to the protected page with authorization credentials that can be verified by the agent. The agent allows the user authorized by AM to access the page.

You can use AM to protect physical devices connected on the Internet of things (IoT). For example, a delivery van tracking system could have its proxying gateway authenticate to a brokering system using an X.509 certificate to allow it to enable an HTTPS protocol and then connect to sensors in its delivery trucks. If the X.509 certificate is valid, the brokering system can monitor a van's fuel consumption, speed, mileage, and overall engine condition to maximize each van's operating efficiency.

Authentication Features

AM supports a number of authentication features and services for use in your deployment:

- **Authentication Nodes and Trees**. AM provides a number of authentication nodes to handle different modes of authenticating users. The nodes must be connected together in a *tree* to provide multiple authentication paths to users. For more information, see "About Authentication Trees".

- **Authentication Modules and Chains**. AM provides a number of authentication modules to handle different modes of authenticating users or entities. The modules also can be *chained* together to provide multiple authentication mechanisms, so that a user's or entity's credentials must be evaluated by one module before control passes to another module. For more information, see "About Authentication Modules and Chains".

- **Authentication Levels**. AM allows each module to be configured with an *authentication level*, which indicates the security level of the user's or entity's credentials. If the user needs to gain access to more sensitive resources, AM may require the user or entity to reauthenticate, providing an additional credential of another type. For more information, see "About Authentication Levels".

- **Social Authentication**. You can configure AM to accept authentication provided by popular third-party identity providers, such as Facebook, Google, and Microsoft. For more information, see "About Social Authentication".

- **Multi-Factor Authentication**. AM supports multi-factor authentication, which requires a user to provide multiple forms of credentials, such as username and password, and a one-time password sent to a user's mobile phone. For more information, see "About Multi-Factor Authentication".

- **Account Lockout**. AM can lock accounts after a pre-configured number of failed authentication attempts. Account lockout works with modules for which users enter a password. For more information, see "About Account Lockout".

- **Sessions**. AM creates an `authentication session` to track the progress of a user or entity as they authenticate. Once the user is authenticated, AM creates a `session` to identify the user or entity during any flow.

  For more information, see "About Sessions".

- **Single Sign-On**. AM allows a user or an entity to use one set of credentials to access multiple applications within a single domain. This is known as single sign-on (SSO). AM also supports Cross-Domain Single Sign-On (CDSSO). For more information, see "About Single Sign-On".

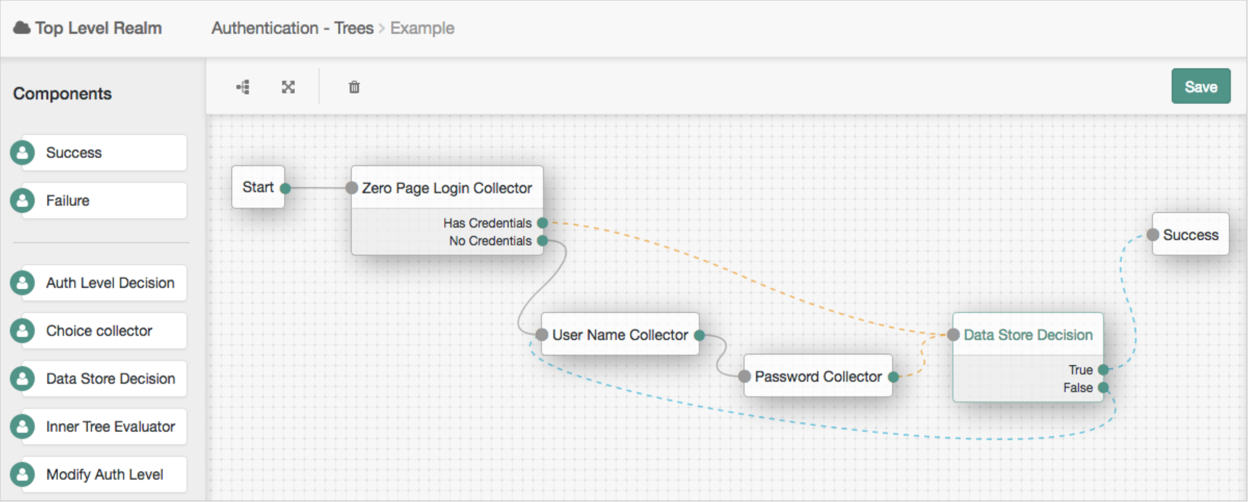
About Authentication Trees

Authentication trees (also referred to as Intelligent Authentication) provide fine-grained authentication by allowing multiple paths and decision points throughout the authentication flow.

Authentication trees are made up of *authentication nodes*, which define actions taken during authentication, similar to authentication modules within chains. Authentication nodes are more granular than modules, with each node performing a single task such as collecting a username or making a simple decision. Unlike authentication modules, authentication nodes can have multiple outcomes rather than just success or failure.

You can create complex yet customer-friendly authentication experiences by linking nodes together, creating loops, and nesting nodes within a tree.

Example Authentication Tree



The authentication trees technology demonstrates the direction AM is taking for administration of complex authorization scenarios, while offering a streamlined login experience to users.

Authentication trees differ in the following ways to traditional authentication chains:

- Authentication nodes are not yet available for all the functionality provided by authentication modules.
- Authentication trees cannot mix with authentication chains. Each authentication to AM can use either a tree or a chain, but not both together.
- The functionality derived from post-authentication plugins, used traditionally with authentication chains, is handled differently when using trees. For example:

- Session property management is handled by individual nodes. See "Set Session Properties Node".
- Calling out to third-party systems is handled by scripted nodes. See "Scripted Decision Node".
- Registering events to make HTTP POST calls to a server is handled by webhooks. See "Configuring Authentication Webhooks". Note that post-authentication plugins do not get triggered when authenticating to a tree, only to a chain.

About Authentication Modules and Chains

AM allows you to configure authentication processes and then customize how they are applied. AM uses *authentication modules* to handle different ways of authenticating. Basically, each authentication module handles one way of obtaining and verifying credentials. You can chain different authentication modules together. In AM, this is called *authentication chaining*. Each authentication module can be configured to specify the continuation and failure semantics with one of the following four criteria: requisite, sufficient, required, or optional.

Authentication modules in a chain can assign a *pass* or *fail* flag to the authorization request. To successfully complete an authentication chain at least one pass flag must have been achieved, and there must be no fail flags.

Flags are assigned when completing a module as shown in the table below:

Authentication Criteria, Flags, and Continuation Semantics

Criteria	**Fail**	**Pass**	**Example**
Requisite	Assigns fail flag. 🚩 Exits chain.	Assigns pass flag. 🚩 Continues chain.	Active Directory, Data Store, and LDAP authentication modules are often set as requisite because of a subsequent requirement in the chain to identify the user. For example, the Device ID (Match) authentication module needs a user's ID before it can retrieve information about the user's devices.
Sufficient	Assigns no flag. Continues chain.	Assigns pass flag. 🚩 Exits chain.	You could set Windows Desktop SSO as sufficient, so authenticated Windows users are let through, whereas web users must traverse another authentication module, such as one requiring a username and password. One exception is that if you pass a sufficient module after having failed a required module, you will continue through the chain and *will not* exit at that point. Consider using a requisite module instead of a required module in this situation.
Required	Assigns fail flag. 🚩	Assigns pass flag.	You could use a required module for login with email and password, so that it can fail through to another module to handle new users who have not yet signed up.

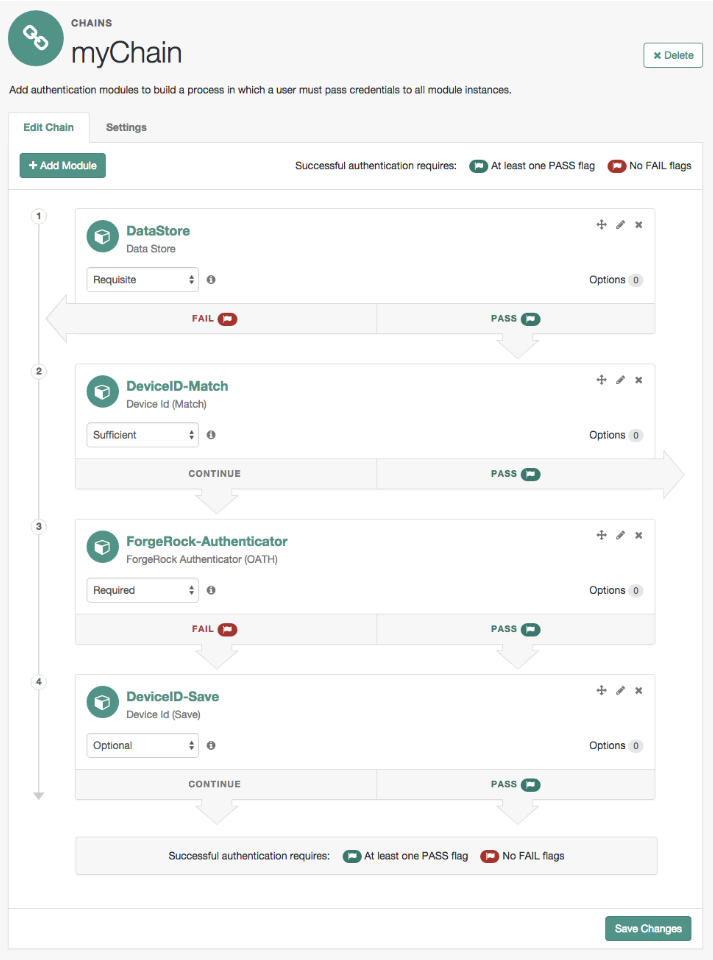
Criteria	Fail	Pass	Example
	Continues chain.	⚑ Continues chain.	
Optional	Assigns no flag. Continues chain.	Assigns pass flag. ⚑ Continues chain.	You could use an optional module to assign a higher authentication level if it passes. Consider a chain with a requisite Data Store module and an optional Certificate module. Users who only passed the Data Store module could be assigned a lower authentication level than users who passed both the Data Store and Certificate modules. The users with the higher authentication level could be granted access to more highly-secured resources.

Tip

In authentication chains with a single module, requisite and required are equivalent. For authentication chains with multiple modules, use required only when you want the authentication chain to continue evaluating modules even after the required criterion fails.

The AM authentication chain editor displays the flags that could be assigned by each module in the chain, and whether execution of the chain continues downwards through the chain or exits out, as shown below:

Authentication Chain with Each Criteria



Add authentication modules to build a process in which a user must pass credentials to all module instances.

Successful authentication requires: At least one PASS flag No FAIL flags

1 DataStore
Data Store
Requisite
Options 0

2 DeviceID-Match
Device Id (Match)
Sufficient
Options 0

3 ForgeRock-Authenticator
ForgeRock Authenticator (OATH)
Required
Options 0

4 DeviceID-Save
Device Id (Save)
Optional
Options 0

Successful authentication requires: At least one PASS flag No FAIL flags

With AM, you can further set *authentication levels* per module, with higher levels being used typically to allow access to more restricted resources. The AM SPIs also let you develop your own authentication modules, and post authentication plugins. Client applications can specify the authentication level, module, user, and authentication service to use among those you have configured. As described later in this guide, you can use *realms* to organize which authentication process applies for different applications or different domains, perhaps managed by different people.

AM leaves the authentication process flexible so that you can adapt how it works to your situation. Although at first the number of choices can seem daunting, now that you understand the basic process, you begin to see how choosing authentication modules and arranging them in authentication chains lets you use AM to protect access to a wide range of applications used in your organization.

About Authentication Levels

When a user successfully authenticates, AM creates a session, which allows AM to manage the user's access to resources. The session is assigned an *authentication level*, which is calculated to be the highest authentication level of any authentication module that passed. If the user's session does not have the appropriate authentication level, then the user may need to reauthenticate again at a higher authentication level to access the requested resource.

The authentication level sets the level of security associated with a module or a tree path. Typically, the strongest form of authentication is assigned the highest authentication level.

If an authentication chain contains `requisite` or `required` modules that were not executed due to the presence of a passing `sufficient` module in front of them, the session's authentication level is calculated to be whichever is greater: the highest authentication level of any authentication module that passed, or the highest authentication level of `requisite` or `required` modules that were not executed.

You can modify AM's default behavior, so that a session's authentication level is *always* the highest authentication level of any authentication module that passed, even if there are `requisite` or `required` modules in the authentication chain that were not executed.

To modify the default behavior, set the `org.forgerock.openam.authLevel.excludeRequiredOrRequisite` property to `true` under Deployment > Servers > *Server Name* > Advanced and restart the AM server.

Authorization policies may also require a particular authentication level to access sensitive resources, which might be at a higher level than the user currently has in the session. When an authenticated user tries to access a sensitive resource with a valid session that does not have the required authentication level, AM returns an authorization decision that denies access to the resource and returns an *advice* indicating that the user needs to reauthenticate at the required authentication level to access the resource. The web or Java agent or policy enforcement point can then send the user back to AM for *session upgrade*. For more information, see"Session Upgrade".

About Social Authentication

AM supports delegated authentication through third-party identity providers, such as Facebook, Google, and VKontakte. The AM console provides setup wizards to configure authentication with each identity provider. An additional wizard provides the ability to configure other third-party authenticators. The following table summarizes the social authentication providers and standards that AM 6.5 supports:

Provider/Standard	Authentication Node?	Authentication Module?	Module Wizard?
OpenID Connect 1.0	Yes	Yes	Yes

Provider/Standard	Authentication Node?	Authentication Module?	Module Wizard?
OAuth 2.0	Yes	Yes	No
Facebook	Yes	Yes [a]	Yes
Google	Yes	Yes [a]	Yes
Instagram	No	Yes	No
Microsoft	Yes [b]	Yes [c]	No
VKontakte	No	Yes	Yes
WeChat	No	Yes	No
WeChat Mobile	No	Yes	No

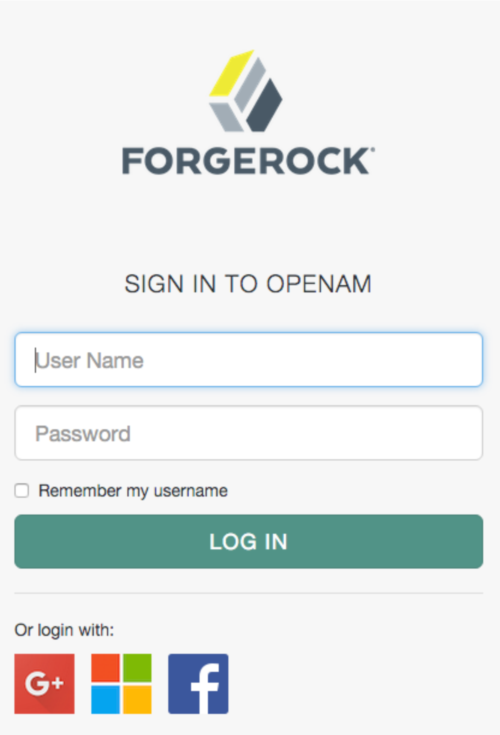
[a] Configure a Social Auth OpenID authentication module.

[b] Configure an OAuth 2.0 authentication node.

[c] Configure a Social Auth OAuth2 authentication module.

Each AM wizard creates an authentication module and an authentication chain containing the correct configuration needed to authenticate with the third party. The wizard also adds configuration data to the realm's *Social Authentication Implementations Service* and provisions the service to enable the display of logos of the configured third-party authentication providers on the AM login screen, as shown below.

Login Screen With Social Authentication Logos



FORGEROCK

SIGN IN TO OPENAM

User Name

Password

□ Remember my username

LOG IN

Or login with:

"*Implementing Social Authentication*" describes how to set up social authentication in AM.

About Multi-Factor Authentication

Multi-factor authentication is an authentication technique that requires users to provide multiple forms of identification when logging in to AM.

This section describes multi-factor authentication features in AM. See "*Implementing Multi-Factor Authentication*" for information about how to set up multi-factor authentication in AM.

Multi-factor authentication provides a more secure method for users to access their accounts with the help of a *device*. Note that the word *device* is used in this section to mean a piece of equipment that can display a one-time password or that supports push notifications using protocols supported by AM multi-factor authentication. Devices are most commonly mobile phones with authenticator apps that support the OATH protocol or push notifications, but could also include other equipment.

The following is an example scenario of multi-factor authentication in AM:

1. An AM administrator configures an authentication chain with the Data Store and ForgeRock Authenticator (OATH) authentication modules.
2. An end user authenticates to AM using that authentication chain.
3. AM prompts the user to enter the user ID and password as required by the Data Store authentication module—the first factor in multi-factor authentication.
4. If the user ID and password were correct, AM prompts the user to obtain a one-time password.
5. The user runs an authenticator app on a mobile phone that generates and displays a one-time password.
6. The user provides the one-time password to AM to successfully complete authentication—the second factor in multi-factor authentication.

Administrators set up multi-factor authentication by creating authentication chains with two or more authentication modules. The initial module in the chain defines the first authentication module for multi-factor authentication. In the preceding scenario, the first authentication module is the Data Store authentication module. Subsequent modules in the chain define the additional factors required to log in, for example the ForgeRock Authenticator (OATH) or ForgeRock Authenticator (Push) authentication modules.

AM supports the Open AuTHentication (OATH) protocols, push notification, and web authentication protocols for multi-factor authentication.

About Open AuTHentication (OATH)

The ForgeRock Authenticator (OATH) module supports HMAC one-time password (HOTP) and time-based one-time password (TOTP) authentication as defined in the OATH standard protocols for HOTP (RFC 4226) and TOTP (RFC 6238). Both HOTP and TOTP authentication require an OATH-compliant device that can provide the password.

HOTP authentication generates the one-time password every time the user requests a new password on their device. The device tracks the number of times the user requests a new one-time password with a counter. The one-time password displays for a period of time you designate in the setup, so the user may be further in the counter on their device than on their account.

AM will resynchronize the counter when the user finally logs in. To accommodate this, you set the number of passwords a user can generate before their device cannot be resynchronized. For example, if you set the number of HOTP Window Size to 50 and someone presses the button 30 times on the user's device to generate a new password, the counter in AM will review the passwords until it reaches the one-time password entered by the user. If someone presses the button 51 times, you will need to reset the counter to match the number on the device's counter before the user can login to AM. HOTP authentication does not check earlier passwords, so if the user attempts to reset the counter on their device, they will not be able to login until you reset the counter in AM to match their device. For more information, see "Resetting Registered Devices by using REST".

TOTP authentication constantly generates a new one-time password based on a time interval you specify. The device tracks the last several passwords generated and the current password. The

TOTP Time Steps setting configures the number of passwords tracked. The Last Login Time setting monitors the time when a user logs in to make sure that user is not logged in several times within the present time period. The TOTP Time-Step Interval should not be so long as to lock users out, with a recommended time of 30 seconds.

Differences Among Authentication Modules That Support HOTP

The ForgeRock Authenticator (OATH), OATH, and HOTP authentication modules let you configure authentication that prompts users to enter HMAC one-time passwords. It is important that administrators understand the differences among these authentication modules:

- The ForgeRock Authenticator (OATH) and OATH authentication modules accept one-time passwords generated by the end user's device, while the HOTP authentication module generates passwords and sends them to users by e-mail or SMS.
- All three of the authentication modules support HOTP passwords. The ForgeRock Authenticator (OATH) and OATH authentication modules also support TOTP passwords.
- The ForgeRock Authenticator (OATH) and OATH authentication modules require users to register their devices, and store the device registration details in the user profile. The HOTP authentication module requires the presence of mobile phone numbers and/or e-mail addresses in user profiles.
- The ForgeRock Authenticator (OATH) authentication module can encrypt stored device registration details.

Before deciding on an implementation strategy, assess your requirements against the following capabilities in AM:

Comparing the ForgeRock Authenticator (OATH) to the HOTP Authentication Module

Requirement	Available With the ForgeRock Authenticator (OATH) Authentication Module?	Available With the HOTP Authentication Module?
End users can authenticate using a HOTP password	Yes	Yes
AM can generate a HOTP password and send it to end users in a text message or an e-mail	No	Yes
End users can register a mobile phone with AM, and an authenticator app on the phone can generate a HOTP or TOTP password that AM accepts as proof of authentication	Yes	No
End users can authenticate with a TOTP password	Yes	No
End users can opt out of providing a one-time password	Yes	No
End users can authenticate using XUI	Yes	Yes

Requirement	Available With the ForgeRock Authenticator (OATH) Authentication Module?	Available With the HOTP Authentication Module?
End users can authenticate using the legacy UI	No	Yes

About Web Authentication (WebAuthn)

Web Authentication allows users to authenticate by using an authenticator device, for example the fingerprint scanner on their laptop or phone.

Communication with the authentication devices is handled by the user's browser. AM requests that the browser activates authenticators with certain criteria; for example it must be built-in to the platform rather than a roaming USB device, and/or that it must verify the identity of the user, rather than simply that a user is present.

To use WebAuthn with AM, users must first register their authenticators. If recovery codes are enabled, users must also make a copy of their codes.

Registration involves the selected authenticator creating, or *minting*, a key pair. The public key of the pair is returned to AM and stored in the user's profile. The private key is stored securely, either in the authenticator itself, or in the platform managing the authenticators. The private key does not leave the client at any time.

When authenticating by using WebAuthn, the authenticator locks some data using the stored private key, which is sent to AM to verify using the public key stored in the user's profile. If the data is verified as being from the correct device, and passes any attestation checks, the authentication is considered successful.

AM supports web authentication in the following user agents and platform minimum versions:

Minimum Web Authentication User Agent Versions

User Agent	Platform	Version	Supported?
Google Chrome	Desktop	70	✓
	Android	70	✓
Microsoft Edge	Desktop	18	✓
Mozilla Firefox	Desktop	60	✓

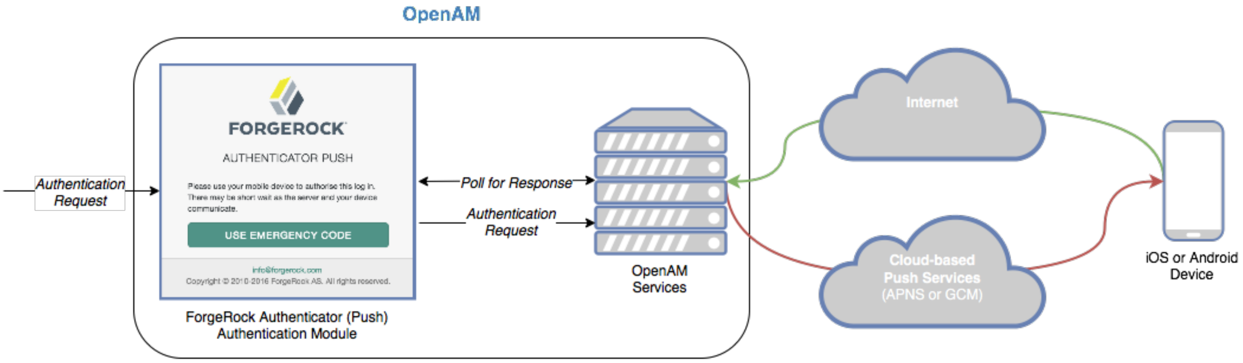
For information on implementing web authentication in AM, see "*Implementing Multi-Factor Authentication*".

About Push Authentication

You can use push notifications as part of the authentication process in AM.

To receive push notifications when authenticating, end users must register an Android or iOS device with AM. The registered device can then be used as an additional factor when authenticating to AM. AM can send the device a push notification, which can be accepted by the ForgeRock Authenticator app. In the app, the user can allow or deny the request that generated the push notification and return the response to AM.

Overview of Push Authentication


OpenAM
Authentication Request
FORGEROCK
AUTHENTICATOR PUSH
Please use your mobile device to authorise this log in. There may be short wait as the server and your device communicate.
USE EMERGENCY CODE
info@forgerock.com
Copyright © 2010-2016 ForgeRock AS. All rights reserved.
ForgeRock Authenticator (Push) Authentication Module
Poll for Response
Authentication Request
OpenAM Services
Internet
Cloud-based Push Services (APNS or GCM)
iOS or Android Device

The following steps occur when AM receives an authentication request and is configured for multi-factor authentication using push notifications:

1. The user must provide credentials to enable AM to locate the user in the identity store and determine if they have a registered mobile device.

2. AM prompts the user to register a mobile device if they have not done so already. Registering a device associates metadata about the device essential for enabling push notifications with the user's profile in the identity store.

   For more information, see"Managing Devices for Multi-Factor Authentication".

3. Once the details of the registered device are obtained, AM creates a push message specific to the registered device. The message has a unique ID, which AM stores in anticipation of a response from the registered device.

   A pending record using the same message ID is also written to the CTS store, providing redundancy should an individual server go offline during the authentication process.

4. AM sends the push message to the registered device.

   AM uses cloud-based push notification services to deliver the messages to the devices. Depending on the registered device, AM uses either Apple Push Notification Services (APNS) or Google Cloud Messaging (GCM) to deliver the push notification.

   The ForgeRock Authenticator (Push) authentication module begins to poll AM and the CTS for an accepted response from the registered device.

5. The user responds to the notification on the registered device, which will open the ForgeRock Authenticator app. In the ForgeRock Authenticator app, the user approves the authentication request with either a swipe, or by using a fingerprint or face recognition on supported hardware.

   For more information, see"To Perform Authentication using Push Notifications".

   The app returns the response to the AM cluster.

6. AM verifies the message is from the correct registered phone and has not been tampered with, and marks the pending record as accepted if valid.

   The ForgeRock Authenticator (Push) module detects the accepted record and redirects the user to their profile page, completing the authentication.

Limitations When Using Passwordless Push Authentication

The ForgeRock Authenticator (Push) authentication module operates in passwordless mode if not preceded by a Data Store module in an authentication chain. When authenticating using such a chain, the user will be asked to enter their user ID, but not their password. A push notification is then sent to their registered device to complete the authentication by using the ForgeRock Authenticator app.

You should be aware of the following potential limitations before deciding to implement passwordless push authentication:

- Unsolicited push messages could be sent to a user's registered device by anyone who knew or was able to guess their user ID.

- If a malicious user attempted to authenticate by using push at the same time as a legitimate user, the legitimate user might unintentionally approve the malicious attempt. This is because push notifications only contain the username and issuer in the text, and it is not easy to determine which notification relates to which authentication attempt.

Consider using push notifications as part of a multi-factor authentication chain For an example, see "Creating Authentication Chains for Push Authentication".

About Account Lockout

AM can lock accounts after repeated authentication failures. Account lockout works with modules for which users can enter a password incorrectly.

"*Implementing Account Lockout*" describes how to set up account lockout in AM.

About Sessions

AM creates an authentication session to track the user's authentication progress through an authentication chain or tree. Once the user has authenticated, AM creates a session to manage the user's or entity's access to resources.

AM session-related services are stateless unless otherwise indicated; they do not hold any session information local to the AM instances. Instead, they store session information either in the CTS token store or on the client. This architecture allows you to scale your AM infrastructure horizontally since any server in the deployment can satisfy any session's request.

Sessions are highly configurable. We recommend you become familiar with basic session concepts before attempting to configure sessions for your environment:

- Sessions have different characteristics depending on where AM stores the sessions. For more information, see "Session Storage Location".
- Sessions require the user or client to be able to hold on to cookies. Cookies provided by AM's Session Service may contain a JSON Web Token (JWT) with the session or just a reference to where the session is stored. For more information, see "Session Cookies".
- Session termination effectively logs the user or entity out of all realms, but the way AM terminates sessions is different depending on where AM stores the sessions. For more information, see "Session Termination".
- Sessions can be upgraded to provide access to sensitive resources. For more information, see "Session Upgrade".
- Sessions can be configured to limit the number of active sessions a user can have at a given time. For more information, see"Session Quotas".
- Sessions can store custom information using post-authentication plugins. For more information about post-authentication plugins, see"Creating Post-Authentication Plugins for Chains".

Session Storage Location

Session storage location is configured at the realm level. The following table illustrates where AM can store sessions:

Session Storage Location

	In the CTS Token Store	On the Client	In AM's Memory
Authentication Sessions	✔ [a]	✔ [a] (Default in new installations)	✔ [b] (Default after upgrade)
Sessions	✔ (Default)	✔	✖

[a] Authentication trees only.

[b] Available for authentication trees and authentication chains.

Session storage location can be heterogeneous within the same AM deployment to suit the requirements of each of your realms.

CTS-Based Sessions

CTS-based sessions reside in the CTS token store and can be cached in memory on one or more AM servers to improve system performance [1] . If the session request is redirected to an AM server that does not have the session cached, that server must retrieve the session from the CTS token store.

AM sends a reference to the session to the client, but the reference does not contain any of the session state information. AM can modify a session during its lifetime without changing the client's reference to the session.

- **CTS-Based Authentication Sessions Specifics**

  CTS-based authentication sessions are *supported for authentication trees only.*

  During authentication, the session reference is returned to the client after a call to the `authenticate` endpoint and stored in the `authId` object of the JSON response.

  AM maintains the authenticating user's session in the CTS token store. After the authentication flow has completed, if the realm to which the user has authenticated is configured for client-based sessions, AM returns session state to the client and deletes the CTS-based session.

  Authentication session whitelisting is an optional feature that maintains a list of in-progress authentication sessions and their progress in the authentication flow to protect against replay attacks. For more information, see"Implementing Authentication Session Whitelisting".

- **CTS-Based Sessions**

  Once the user is authenticated, the session reference is known as an *SSO token*. For browser clients, AM sets a cookie in the browser that contains the session reference. For REST clients, AM returns the session reference in response to calls to the `authentication` endpoint.

  For more information about session cookies, see "Session Cookies".

Client-Based Sessions

Client-based sessions are those where AM returns session state to the client after each request, and require it to be passed in with the subsequent request.

[1] For information about configuring AM with sticky load balancing, see "Configuring Load Balancing for a Site" in the *Installation Guide*.

Important

Some features are not supported in realms configured for client-based sessions. For more information, see "Limitations When Using Client-Based Sessions".

You should configure AM to sign and/or encrypt client-based sessions and authentication sessions for security reasons. As decrypting and verifying the session may be an expensive operation to perform on each request, AM caches [1] the decrypt sequence in memory to improve performance.

For more information about configuring client-based security, see "Configuring Client-Based Session and Authentication Session Security".

- **Client-Based Authentication Sessions Specifics**

  Client-based authentication sessions are *supported for authentication trees only*, and are configured by default in new installations.

  During authentication, authentication session state is returned to the client after each call to the `authenticate` endpoint and stored in the `authId` object of the JSON response.

  After the authentication flow has completed, if the realm to which the user has authenticated is configured for CTS-based sessions, AM creates the user's session in the CTS token store. Then, AM attempts to invalidate the client-based authentication session.

  Storing authentication sessions on the client allows any AM server to handle the authentication flow at any point in time without load-balancing requirements.

  Authentication session whitelisting is an optional feature that maintains a list of in-progress authentication sessions and their progress in the authentication flow to protect against replay attacks. For more information, see"Implementing Authentication Session Whitelisting".

- **Client-Based Sessions Specifics**

  For browser-based clients, AM sets a cookie in the browser that contains the session state. When the browser transmits the cookie back to AM, AM decodes the session state from the cookie. For REST-based clients, AM sends the cookie in a header. For more information about session cookies, see "Session Cookies".

  Session blacklisting is an optional feature that maintains a list of logged out client-based sessions in the CTS token store. For more information about session termination and session blacklisting, see "Session Termination" and "Configuring Session Blacklisting".

In-Memory Sessions

In-memory sessions reside in AM's memory. AM sends clients a reference to the session, but the reference does not contain any of the session state information.

- **In-Memory Authentication Sessions Specifics**

In-memory authentication sessions are the *only configuration supported for authentication chains*. They are also configured by default for authentication trees after an upgrade.

During authentication, the authentication session reference is returned to the client after a call to the `authenticate` endpoint and stored in the `authId` object of the JSON response.

AM maintains the user's authentication session in its memory. After the authentication flow has completed, AM performs the following tasks:

- If the realm to which the user has authenticated is configured for CTS-based sessions, AM stores the user's session in the CTS token store and deletes the authentication session from memory.
- If the realm to which the user has authenticated is configured for client-based sessions, AM stores the user's session in a cookie on the user's browser and deletes the authentication session from memory.

Authentication session whitelisting is an optional feature that maintains a list of in-progress authentication sessions and their progress in the authentication flow to protect against replay attacks. For more information, see"Implementing Authentication Session Whitelisting".

> **Important**
>
> Deployments where AM stores authentication sessions in memory require sticky load balancing to route all requests pertaining to a particular authentication flow to the same AM server. If a request reaches a different AM server, the authentication flow will start anew.
>
> Authentication chains only support storing authentication sessions in memory. ForgeRock recommends switching to authentication trees with CTS-based or client-based authentication sessions.
>
> For information about configuring AM with sticky load balancing, see "Configuring Load Balancing for a Site" in the *Installation Guide*.

- **In-Memory Sessions Specifics**

  AM does not support in-memory sessions for authenticated users.

Choosing Where to Store Sessions

AM stores CTS-based sessions in the CTS token store and caches sessions in server memory. If a server with cached sessions fails, or if the load balancer in front of AM servers directs a request to a server that does not have the user's session cached, the AM server retrieves the session from the CTS token store, incurring performance overhead.

Choosing where to store sessions is an important decision you must make by realm. Consider the information in the following tables before configuring sessions:

- "Advantages of CTS-Based Sessions"

- "Advantages of Client-Based Sessions"
- "Advantages of In-Memory Sessions"
- "Impact of Storage Location for Authentication Sessions"
- "Impact of Storage Location for Sessions"

CTS-based sessions provide the following advantages:

Advantages of CTS-Based Sessions

Advantage	Applies to Authentication Sessions?	Applies to Sessions?
Full Feature Support CTS-based sessions support all AM features, such as CDSSO and quotas. Client-based sessions do not. For information about restrictions on AM usage with client-based sessions, see "Limitations When Using Client-Based Sessions". This advantage does not apply to authentication sessions, since they do not provide features.	–	✓
Session Information Is Not Resident In Browser Cookies With CTS-based sessions, all the information about the session resides in CTS and might be cached on one or more AM servers. With client-based sessions, session information is held in browser cookies. This information could be very long-lived.	✓	✓

Client-based sessions provide the following advantages:

Advantages of Client-Based Sessions

Advantage	Applies to Authentication Sessions?	Applies to Sessions?
Unlimited Horizontal Scalability for Session Infrastructure Client-based sessions provides unlimited horizontal scalability for your sessions by storing the session state on the client as a signed and encrypted JWT. Overall performance on hosts using client-based sessions can be easily improved by adding more hosts to the AM deployment.	✓	✓
Replication-Free Deployments Global deployments may struggle to keep their CTS token store replication in sync when distances are long and updates are frequent.	✓	✓

Advantage	Applies to Authentication Sessions?	Applies to Sessions?
Client-based sessions are not constrained by the replication speed of the CTS token store. Therefore, client-based sessions are usually more suitable for deployments where a session can be serviced at any time by any server.		

In-memory authentication sessions provide the following advantages:

Advantages of In-Memory Sessions

Advantage	Applies to Authentication Sessions?	Applies to Sessions?
Faster Performance With Equivalent Host AM servers configured for in-memory authentication sessions can validate more sessions per second per host than those configured for client-based or CTS-based authentication sessions.	✔	✖
Session Information Is Not Resident in Browser Cookies Authentication session information resides in AM's memory and it is not accessible to users. With client-based sessions, authentication session information is held in browser cookies.	✔	✖

The following table contrasts the impact of storing authentication sessions in different locations:

Impact of Storage Location for Authentication Sessions

	CTS-Based Authentication Sessions	Client-Based Authentication Sessions	In-Memory Authentication Sessions
Authentication Method	Authentication trees.	Authentication trees.	Authentication trees and authentication chains.
Session Location	Authoritative source: CTS token store. Sessions might also be cached in AM's memory for improved performance.	On the client.	In AM server's memory.
Load Balancer Requirements	None. Session stickiness recommended for performance.	None. Session stickiness recommended for performance.	Session stickiness.
Core Token Service Usage	Authoritative source for user sessions. Session whitelisting, when enabled.	Session whitelisting, when enabled.	None.
Uninterrupted Session Availability	No special configuration required.	No special configuration required.	Not available.

	CTS-Based Authentication Sessions	Client-Based Authentication Sessions	In-Memory Authentication Sessions
Session Security	Sessions reside in the CTS token store, and are not accessible to users.	Sessions reside on the client and should be signed and encrypted.	Sessions reside in AM's memory, and are not accessible to users.

The following table contrasts the impact of storing sessions in different locations:

Impact of Storage Location for Sessions

	CTS-Based Sessions	Client-Based Sessions
Hardware	Higher I/O and memory consumption.	Higher CPU consumption.
Logical Hosts	Variable or large number of hosts.	Variable or large number of hosts.
Session Monitoring	Available.	Not available.
Session Location	Authoritative source: CTS token store. Sessions might also be cached in AM's memory for improved performance.	In a cookie in the client.
Load Balancer Requirements	None. Session stickiness recommended for performance.	None. Session stickiness recommended for performance.
Uninterrupted Session Availability	No special configuration required.	No special configuration required.
Core Token Service Usage	Authoritative source for user sessions.	Provides session blacklisting for logged out sessions.
Core Token Service Demand	Heavier.	Lighter.
Session Security	Sessions reside in the CTS token store, and are not accessible to users.	Sessions should be signed and encrypted. [a]
Cross-Domain Single Sign-On Support	All AM capabilities supported.	Web Agents and Java Agents: Supported without restricted tokens.

[a] Web Agents and Java Agents support either signing or encrypting client-based sessions, but not both. For more information, see "Configure Client-Based Session Security for Agents".

Session Cookies

AM issues a cookie to the user or entity regardless of the session location for client-based and CTS-based sessions. By default, the cookie's name is `iPlanetDirectoryPro`. For sessions stored in the CTS token store, the cookie contains a reference to the session in the CTS token store and several other pieces of information. For sessions stored on the client, the `iPlanetDirectoryPro` cookie contains all the information that would be held in the CTS token store.

Client-based session cookies are comprised of two parts. The first part of the cookie is identical to the cookie used by CTS-based sessions, which ensures the compatibility of the cookies regardless of the session location. The second part is a JSON Web Token (JWT), and it contains session information, as illustrated below:

- `iPlanetDirectoryPro` cookie for CTS-based sessions:

  ```
  AQIC...sswo.*AAJ...MA..*
  ```

- `iPlanetDirectoryPro` cookie for Client-based sessions:

  ```
  AQIC...sswo.*AAJ...MA..*ey...................................fQ.
  ```

Note that the examples are not to scale. The size of the client-based session cookie increases when you customize AM to store additional attributes in users' sessions. You are responsible for ensuring that the size of the cookie does not exceed the maximum cookie size allowed by your end users' browsers.

> **Note**
>
> You should configure AM to sign and/or encrypt client-based sessions to ensure a malicious user cannot tamper with the session or read its contents. For more information, see "Configuring Client-Based Session and Authentication Session Security".

Session Termination

AM manages active sessions, allowing single sign-on when authenticated users attempt to access system resources in AM's control.

AM ensures that user sessions are terminated when a configured timeout is reached, or when AM users perform actions that cause session termination. Session termination effectively logs the user out of all systems protected by AM.

With CTS-based sessions, AM terminates sessions in four situations:

- When a user explicitly logs out
- When an administrator monitoring sessions explicitly terminates a session
- When a session exceeds the maximum time-to-live
- When a user is idle for longer than the maximum session idle time

Under these circumstances, AM responds by removing CTS-based sessions from the CTS token store and from AM server memory caches. With the user's session no longer present in CTS, AM forces the user to reauthenticate during subsequent attempts to access resources protected by AM.

When a user explicitly logs out of AM, AM also attempts to invalidate the `iPlanetDirectoryPro` cookie in users' browsers by sending a `Set-Cookie` header with an invalid session ID and a cookie expiration time that is in the past. In the case of administrator session termination and session timeout, AM cannot invalidate the `iPlanetDirectoryPro` cookie until the next time the user accesses AM.

Session termination differs for client-based sessions. Since client-based sessions are not maintained in the CTS token store, administrators cannot monitor or terminate them. Because AM does not modify the `iPlanetDirectoryPro` cookie for client-based sessions after authentication, the session idle time is not maintained in the cookie. Therefore, AM does not automatically terminate client-based sessions that have exceeded the idle timeout.

As with CTS-based sessions, AM attempts to invalidate the `iPlanetDirectoryPro` cookie from a user's browser when the user logs out. When the maximum session time is exceeded, AM also attempts to invalidate the `iPlanetDirectoryPro` cookie in the user's browser the next time the user accesses AM.

It is important to understand that AM cannot guarantee cookie invalidation. For example, the HTTP response containing the `Set-Cookie` header might be lost. This is not an issue for CTS-based sessions, because a logged out session no longer exists in the CTS token store, and a user who attempts to access AM after previously logging out will be forced to reauthenticate.

However, the lack of a guarantee of cookie invalidation is an issue for deployments with client-based sessions. It could be possible for a logged out user to have an `iPlanetDirectoryPro` cookie. AM could not determine that the user previously logged out. Therefore, AM supports a feature that takes additional action when users log out of client-based sessions. AM can maintain a list of logged out client-based sessions in a session blacklist in the CTS token store. Whenever users attempt to access AM with client-based sessions, AM checks the session blacklist to validate that the user has not, in fact, logged out.

For more information about session blacklist options, see "Configuring Session Blacklisting".

Session Upgrade

Consider a website for a University. Some information, such as courses and degree catalogs, are free for anyone to see and therefore, do not need to be protected. The University also provides the students with a portal they can use to see their grades, which is protected with a policy that requires users to authenticate. However, to pay tuition, students are required to present additional credentials to increase their authentication level and gain access to these functions.

Allowing authenticated users to provide additional credentials to access sensitive resources is called *session upgrade*, which is AM's mechanism to perform step-up authentication.

Session upgrade triggers during the following events:

- An authenticated user is redirected to a URL that has the `ForceAuth` parameter set to `true`. For example, `https://openam.example.com:8443/openam/XUI/?realm=/myRealm&ForceAuth=true#login`

  In this case, the user is asked to reauthenticate to the default chain in the realm `myRealm`.

> **Important**
>
> Session upgrade using the `ForceAuth` parameter is only supported for CTS-based sessions.

- An authenticated user tries to access a protected resource and AM sends the policy enforcement point (PEP) *advice* that the user should perform one of the following actions:

  - Authenticate at an authentication level greater than the current level
  - Authenticate to a module
  - Authenticate to a service

  The flow of the session upgrade during policy evaluation is as follows:

  1. An authenticated user tries to access a resource.
  2. The PEP, for example a web or Java agent, sends the request to AM for policy decision.
  3. AM returns an authorization decision that denies access to the resource, and returns an *advice* indicating that the user needs to present additional credentials to access the resource.
  4. The policy enforcement point sends the user back to AM for session upgrade.
  5. The user provides additional credentials. For example, they may provide a one-time password, swipe their phone screen, or use face recognition.
  6. AM authenticates the user.
  7. The user can access now the sensitive resource.

If the realm is configured for CTS-based sessions, one of the following will happen, depending on the mechanism used to perform session upgrade:

- If the realm is configured for CTS-based sessions, one of the following will happen, depending on the mechanism used to perform session upgrade:

  - When using the `ForceAuth` parameter, AM does one of the following:

    - (Authentication trees only) AM issues new session tokens to users on reauthentication, even if the current session already meets the security requirements.
    - (Authentication chains only) AM does not issue new session tokens on reauthentication, regardless of the security level they are authenticating to. Instead, it updates the session token with the new authentication information, if required.

  - When using *advices*, AM copies the session properties to a new session and hands the client a new session token to replace the original one. The new session reflects the successful authentication to a higher level.

- If the realm is configured for client-based sessions, AM hands the client a new session token to replace the original one. The new session reflects the successful authentication to a higher level.

If session upgrade is unsuccessful, AM leaves the user session as it was before the attempt at stronger authentication. If session upgrade failed because the login page times out, AM redirects the user's browser to the success URL from the last successful authentication.

If session upgrade is unsuccessful, AM leaves the user session as it was before the attempt at stronger authentication. If session upgrade failed because the login page times out, AM redirects the user's browser to the success URL from the last successful authentication.

> **Tip**
>
> Anonymous sessions can also be upgraded to non-anonymous sessions by using the "Anonymous Session Upgrade Node".

Session Quotas

In some deployments, you need to limit how many active sessions a user can have at a given time. For example, you might want to prevent a user from using more than two devices at once.

AM lets you limit the number of active sessions for a user by setting session quotas. You also configure session quota exhaustion actions so that when a user goes beyond the session quota, AM takes the appropriate action.

See "Implementing Session Quotas" for instructions.

AM's support for session quotas requires CTS-based sessions. Be sure that AM is configured for CTS-based sessions—the default configuration—before attempting to configure session quotas.

About Single Sign-On

Single sign-on (SSO) lets users who have authenticated to AM access multiple independent services from a single login session by storing user sessions as HTTP cookies[2].

Cross-domain single sign-on (CDSSO) is an AM-specific capability that provides SSO inside the same organization within a single domain or across domains. For example, CDSSO allows your AM servers in the DNS domain `.internal.net` to provide authentication and authorization to web and Java agents from the `.internal.net` domain and other DNS domains, such as `.example.net`.

[2] If you are unfamiliar with HTTP cookies, see "About HTTP Cookies" for more information.

Since CDSSO removes the constraint of configuring SSO depending on the DNS domain, it simplifies the deployment of SSO in your environment.

When implementing CDSSO, take into account the following points:

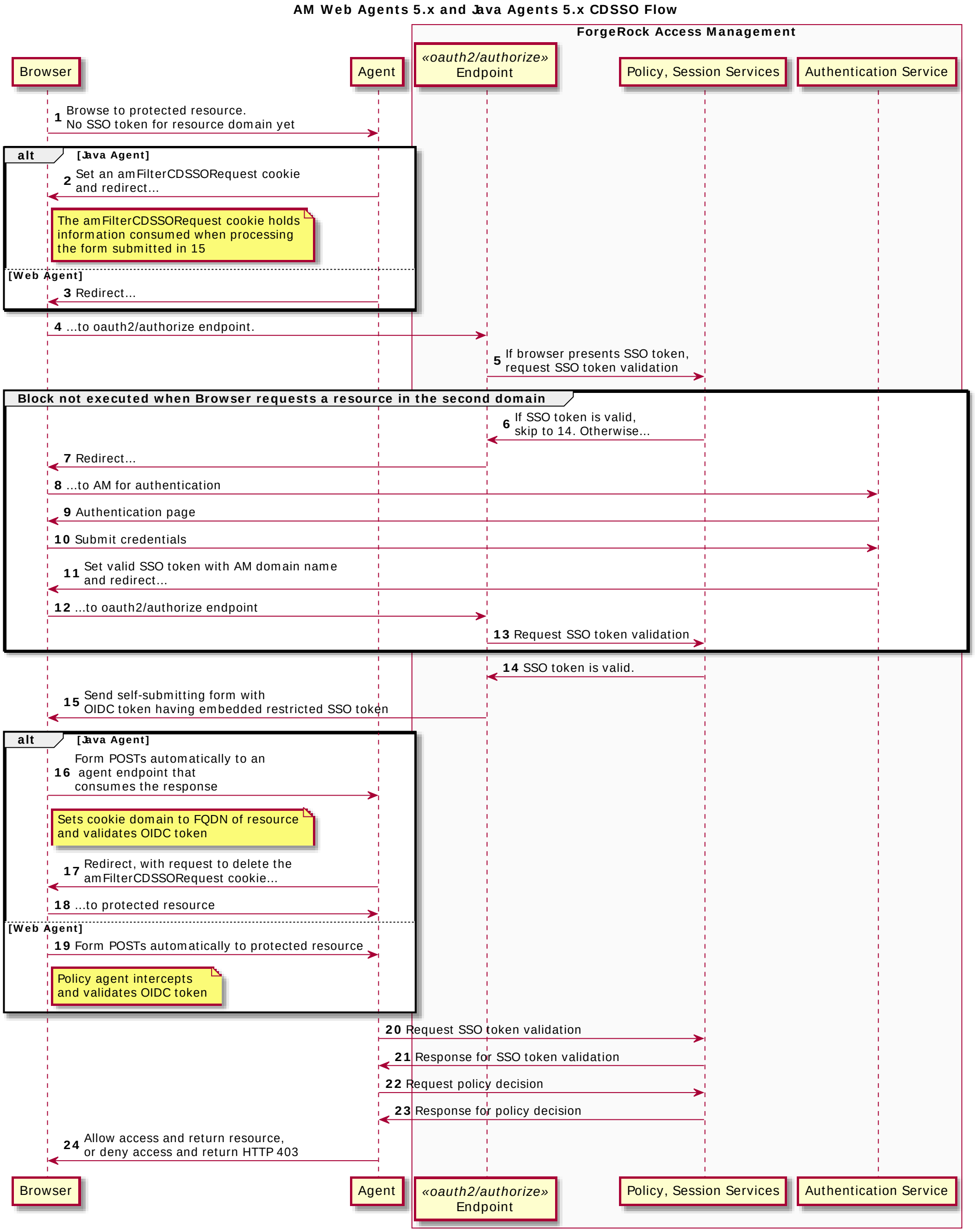
- For SSO across multiple organizations or when integrating with other access management software, use AM's federation capabilities, such as OAuth 2.0 or SAML v2.0.
- Web Agents and Java Agents both support CDSSO.

  AM also supports CDSSO with IG version 6 or later. For more information, see Single Sign-On and Cross-Domain Single Sign-On in the *ForgeRock Identity Gateway Gateway Guide*.
- CDSSO supports CTS-based and client-based sessions. For more information about session state impact on CDSSO, see"Impact of Storage Location for Sessions".

Web Agents and Java Agents wrap the SSO session token inside an OpenID Connect (OIDC) JSON Web Token (JWT). During the CDSSO flow, the agents create cookies for the different domains specified in the agent profile, and the `oauth2/authorize` endpoint authorizes the different cookie domains as required.

The following diagram illustrates the CDSSO flow for Web Agents and Java Agents:

Web and Java Agents CDSSO Flow


AM Web Agents 5.x and Java Agents 5.x CDSSO Flow
ForgeRock Access Management
Browser
Agent
«oauth2/authorize» Endpoint
Policy, Session Services
Authentication Service
1 Browse to protected resource. No SSO token for resource domain yet
alt [Java Agent]
2 Set an amFilterCDSSORequest cookie and redirect...
The amFilterCDSSORequest cookie holds information consumed when processing the form submitted in 15
[Web Agent]
3 Redirect...
4 ...to oauth2/authorize endpoint.
5 If browser presents SSO token, request SSO token validation
Block not executed when Browser requests a resource in the second domain
6 If SSO token is valid, skip to 14. Otherwise...
7 Redirect...
8 ...to AM for authentication
9 Authentication page
10 Submit credentials
11 Set valid SSO token with AM domain name and redirect...
12 ...to oauth2/authorize endpoint
13 Request SSO token validation
14 SSO token is valid.
15 Send self-submitting form with OIDC token having embedded restricted SSO token
alt [Java Agent]
16 Form POSTs automatically to an agent endpoint that consumes the response
Sets cookie domain to FQDN of resource and validates OIDC token
17 Redirect, with request to delete the amFilterCDSSORequest cookie...
18 ...to protected resource
[Web Agent]
19 Form POSTs automatically to protected resource
Policy agent intercepts and validates OIDC token
20 Request SSO token validation
21 Response for SSO token validation
22 Request policy decision
23 Response for policy decision
24 Allow access and return resource, or deny access and return HTTP 403
Browser
Agent
«oauth2/authorize» Endpoint
Policy, Session Services
Authentication Service

For information about how to configure CDSSO, see "Implementing Cross-Domain Single Sign-On".

About Realms and SSO

When changing authentication realms, a subject leaves the current SSO realm. The new SSO realm might apply to different applications, and use a different authentication process. For AM, logging in to a new realm means logging out of the current realm.

When a user interactively changes realms through the AM console, AM offers the option of logging out of the current realm to log in to the new realm, or choosing to remain logged in to the current realm.

The result depends on the user's choice:

- If the user cancels the change at this point, the user remains logged in to the current realm, and is not logged in to the new realm.
- If the user chooses to log in to the new realm, AM first logs the user out of the current realm, and then prompts the user to log in to the new realm.

Chapter 2
Implementing Authentication

AM supports a wide range of authentication modules that can be configured together using authentication chains, and authentication nodes that can be configured together using authentication trees.

AM also supports post-authentication plugins to customize any process after the user or the entity has been authenticated.

After you configure AM authentication, users can authenticate to AM using a browser or a REST API call as described in "*Using Authentication*".

This chapter presents the available authentication modules and nodes, and procedures to configure chains, trees, and post-authentication plugins:

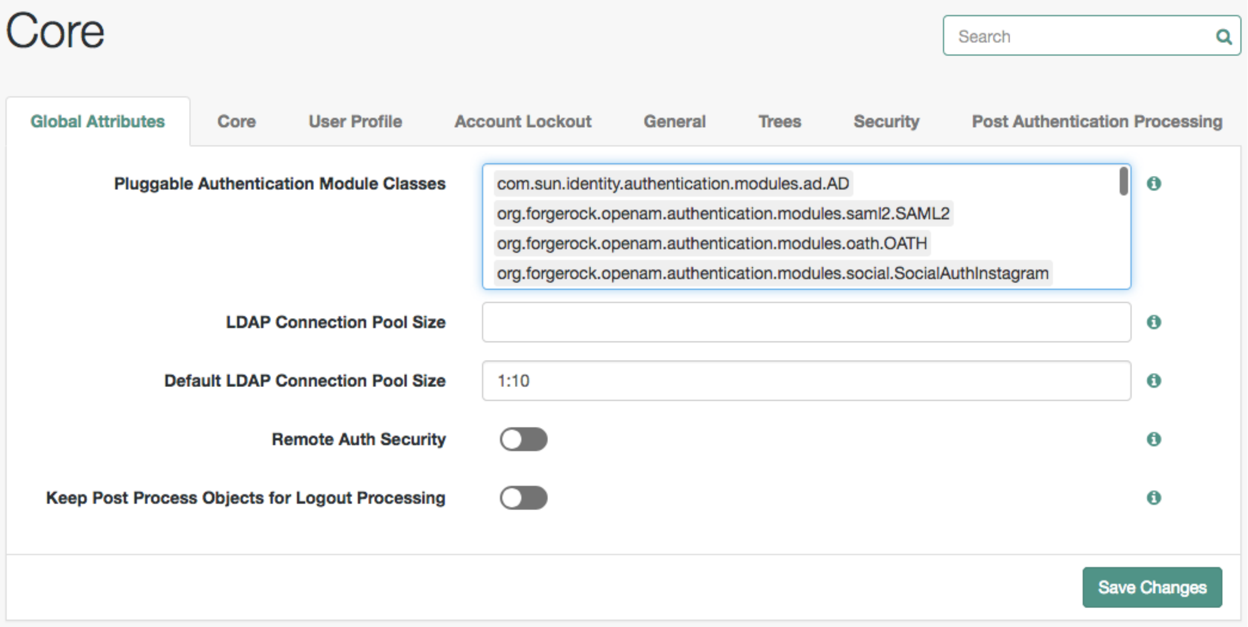
- "Setting up a Realm for Authentication"
- "Configuring Authentication Trees"
- "Configuring Authentication Chains and Modules"
- "Configuring the Default Authentication Tree or Chain"

Setting up a Realm for Authentication

In AM, users always authenticate to a realm. Every AM realm has a set of authentication properties that applies to all authentication performed to that realm. The settings are referred to as *core authentication attributes*.

To configure core authentication attributes for an entire AM deployment, navigate to Configure > Authentication in the AM console, and then click Core Attributes.

The Core Authentication Attributes Page



Pluggable Authentication Module Classes: com.sun.identity.authentication.modules.ad.AD, org.forgerock.openam.authentication.modules.saml2.SAML2, org.forgerock.openam.authentication.modules.oath.OATH, org.forgerock.openam.authentication.modules.social.SocialAuthInstagram

LDAP Connection Pool Size:

Default LDAP Connection Pool Size: 1:10

Remote Auth Security: ○

Keep Post Process Objects for Logout Processing: ○

To override the global core authentication configuration in a realm, navigate to Realms > *Realm Name* > Authentication > Settings in the AM console. Note that when you configure core authentication attributes in a realm, the Global tab does not appear.

Use core authentication attributes to configure:

- The list of available authentication modules
- Which types of clients can authenticate with which modules
- Connection pools for access to directory servers
- Whether to retain objects used during authentication so they can be used at logout
- Defaults for configuring authentication in a particular realm

For detailed information about the core configuration attributes, see "Core Authentication Attributes".

Configuring Authentication Trees

This section covers creating authentication trees and configuring authentication nodes.

Authenticating to a tree uses the same syntax as authenticating to a chain. For example, to authenticate to a tree called `myAuthTree` in the top level realm, use a URL similar to the following: `https://openam.example.com:8443/openam/XUI/?realm=/&service=myAuthTree#login`

To Create an Authentication Tree

1. On the Realms page of the AM console, select the realm in which to create the authentication tree.

2. On the Realm Overview page, select Authentication in the left-hand menu, and then select Trees.

3. On the Trees page, select Create Tree. Enter a tree name, for example `myAuthTree`, and then select Create.

   The authentication tree designer is displayed, with the Start entry point connected to the Failure exit point.

   The authentication tree designer provides the following features on the toolbar:

   Authentication Tree Designer Toolbar

Button	Usage
⊢	Lay out and align nodes according to the order they are connected.
⤢	Toggle the designer window between normal and full screen layout.
🗑	Remove the selected node. Note that the Start entry point cannot be deleted.

4. Add a node to the tree by dragging the node from the Components panel on the left-hand side and dropping it into the designer area.

5. (Optional) Configure the node properties by using the right-hand panel. For more information on the available properties for each node, see "Configuring Authentication Nodes"

6. Connect the node to the tree as follows:

   - Select and drag the output connector from an existing node and drop it onto the new node.

   - Select and drag the output connector from the new node and drop it onto an existing node.

   Nodes have one or more connectors, displayed as dots on the node. Unconnected connectors are colored red and must be connected to other nodes in the tree.

   > **Tip**
   >
   > *Input* connectors appear on the left of the node, *output* connectors appear on the right.

   A line is drawn between the connectors of connected nodes, and the connectors will no longer be red.

7. (Optional) Alter a connection by selecting and dragging the green connector in the connection and dropping it onto the new location.

8. Continue adding, connecting and removing nodes until the tree is complete, and then select Save.

9. Test your authentication tree by navigating to a URL similar to the following: `https://openam.example.com:8443/openam/XUI/?realm=/&service=myAuthTree#login/`

> **Tip**
>
> Clean installs of AM with an embedded data store provide ready-made sample authentication trees to demonstrate how they can be put together. These sample trees are not installed by default if you are upgrading an existing instance of AM. The `sample-trees-6.5.5.zip` file, in the main `AM-6.5.5.zip` download package, contains the sample trees in JSON files, ready for import by *Amster* command-line interface. For information on importing files by using Amster, see Importing Configuration Data in the *Amster 6.5 User Guide*.

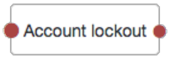
Configuring Authentication Nodes

This section covers the configuration of the authentication nodes that are built into AM.

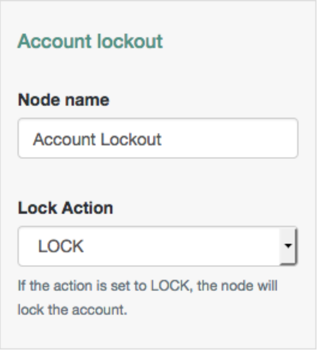
> **Tip**
>
> A number of additional authentication nodes are available from the ForgeRock Marketplace website.

Account lockout Node

The Account lockout node can lock or unlock the authenticating user's account profile.



Properties:



Account lockout

Node name

Account Lockout

Lock Action

LOCK

If the action is set to LOCK, the node will lock the account.

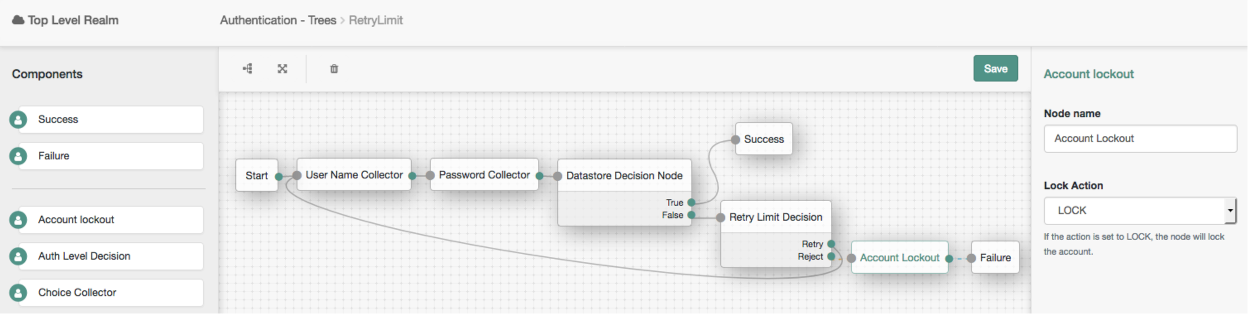
Property	Usage
Lock Action	Choose whether to `LOCK` or `UNLOCK` the authenticating user's account profile.

Property	Usage
	The Data Store Decision authentication node checks if the account profile is in the LOCK state. For more information, see "Data Store Decision Node".

Example:

The following example uses the Account lockout Decision authentication node with the Retry Limit Decision Node to lock an account after a number of invalid attempts:

RetryLimit Tree With Account lockout Decision Node



Account lockout

Node name: Account Lockout

Lock Action: LOCK

If the action is set to LOCK, the node will lock the account.

Agent Data Store Decision Node

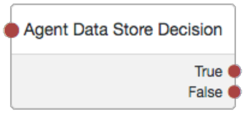
The Agent Data Store Decision authentication node verifies that a provided agent ID and password match a web agent or Java agent profile configured in AM.

Note

Non-agent identities, such as users stored in configured identity repositories, cannot be verified by using the Agent Data Store Decision node. Instead, you should use the Data Store Decision Node.

The web or Java agent ID, and the password should be obtained by using the Zero Page Login Collector Node.

The tree evaluation continues along the `True` path if the credentials match those of a configured agent profile. Otherwise, the tree evaluation continues along the `False` path.



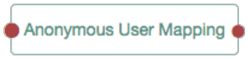
Properties:

This node has no configurable properties.

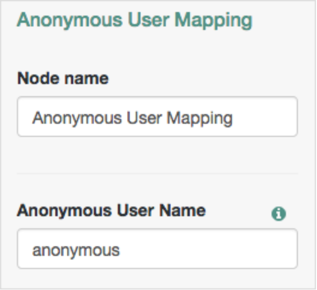
Anonymous User Mapping Node

The Anonymous User Mapping node allows users to log in to your application or web site without providing credentials, by assuming the identity of a specified, existing user account. The default user for this purpose is named `anonymous`.

Typically, you would provide such users with very limited access, for example, anonymous users may have access to public downloads on your site.



Properties:



Anonymous User Mapping

Node name

Anonymous User Mapping

Anonymous User Name

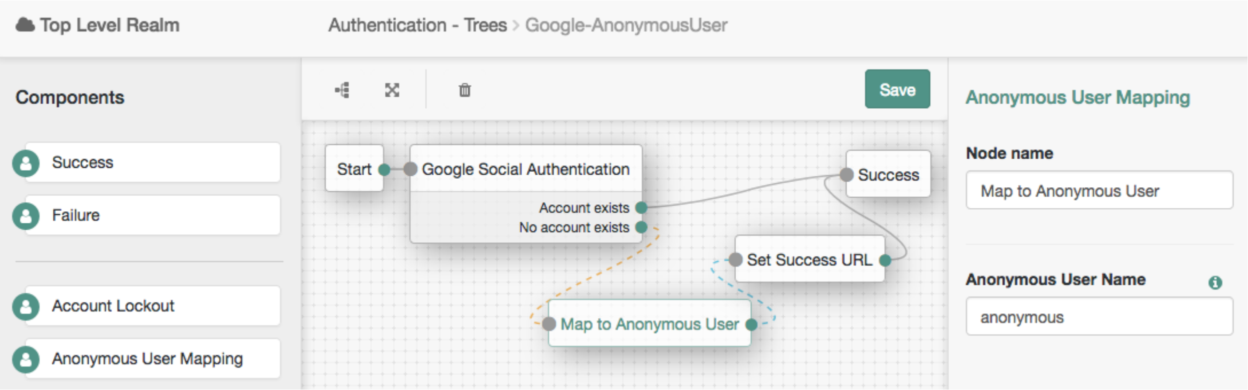
anonymous

Property	Usage
Anonymous User Name	Specifies the username of an account that represents anonymous users. This user must already exist in the realm.

Example:

The following example uses the Anonymous User Mapping authentication node to allow users who have performed social authentication using Google to access AM as an anonymous user if they do not have a matching existing profile.

Google-AnonymousUser Tree With Anonymous User Mapping Node



Top Level Realm

Authentication - Trees > Google-AnonymousUser

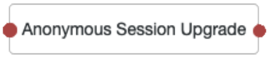
Anonymous User Mapping

Node name: Map to Anonymous User

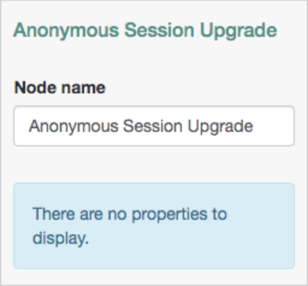
Anonymous User Name: anonymous

Anonymous Session Upgrade Node

The Anonymous Session Upgrade node allows an anonymous session to be upgraded to a non-anonymous session by adding the Anonymous Session Upgrade node as the first node in any tree.



Properties:



Anonymous Session Upgrade

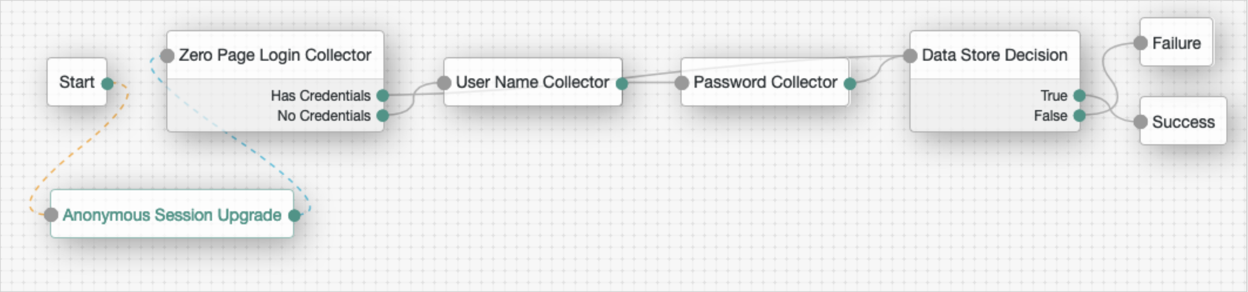
Node name: Anonymous Session Upgrade

There are no properties to display.

Example:

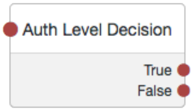
After using the "Anonymous User Mapping Node" to access AM as an anonymous user, the Anonymous Session Upgrade authentication node lets users upgrade their session to a non-anonymous one.

Example Tree With Anonymous Session Upgrade Node

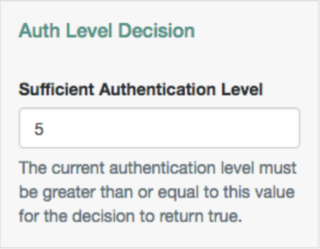


Auth Level Decision Node

The Auth Level Decision authentication node compares the current authentication level value against a configured value.



Properties:



Auth Level Decision

Sufficient Authentication Level

5

The current authentication level must be greater than or equal to this value for the decision to return true.

Property	Usage
Sufficient Authentication Level	The tree evaluation continues along the `True` path if the current authentication level is equal to or greater than the entered integer. Otherwise, the tree evaluation continues along the `False` path.

Choice Collector Node

The Choice Collector authentication node lets you define two or more options to present to the user when authenticating.

Outcomes

- *Choice 1*

  ...

- *Choice n*

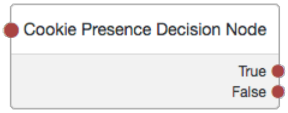
Properties

Property	**Usage**
Choices	Enter two or more choice strings to display to the user. To remove a choice, select its Delete icon (**x**). To delete all choices, select the Clear all button in the Choices field.
Default choice *(required)*	Enter the value of the choice to be selected by default. **Important** If you do not specify a default choice, the first choice in the list becomes the default.
Prompt *(required)*	Enter the prompt string to display to the user when presenting the choices.

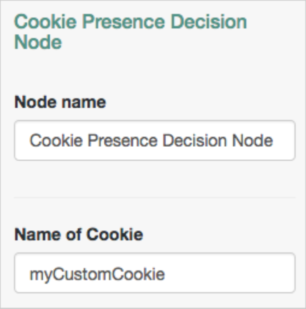
Cookie Presence Decision Node

The Cookie Presence Decision authentication node checks if a named cookie is present in the incoming authentication request.

Note that the node does not check the value of the named cookie, only that it exists.



Properties:



Cookie Presence Decision Node

Node name: Cookie Presence Decision Node

Name of Cookie: myCustomCookie

Property	Usage
Name of Cookie	The tree evaluation continues along the `True` path if the named cookie is present in the incoming authentication request. Otherwise, the tree evaluation continues along the `False` path.

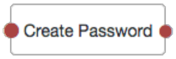
Create Password Node

The Create Password node allows users to create a password when provisioning an account.

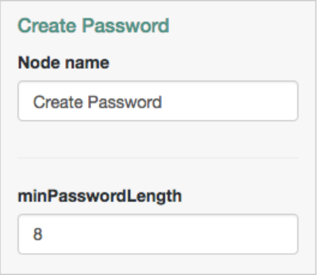
The social identity provider will not provide a user's password. Use this node to provide a password to complete the user's credentials before provisioning an account.

The tree must provision an account after asking the user for a password, for example by using the `Provision Dynamic Account` authentication node. If an account is not provisioned the entered password will not be saved.

> **Note**
>
> You must not place any nodes that request additional input from the user between the Create Password node and the provisioning node, otherwise the password will be lost.



Properties:



Create Password

Node name: Create Password

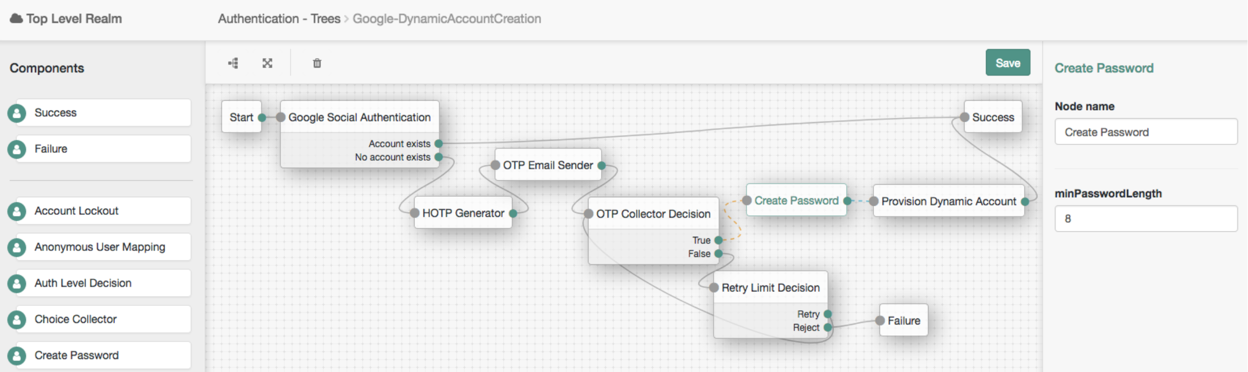
minPasswordLength: 8

Property	Usage
minPasswordLength	Specifies the minimum number of characters the password must contain.

Example:

The following example uses the Create Password authentication node to allow users who have performed social authentication using Google to provide a password and provision an account, if they do not have a matching existing profile. They must enter a one-time password to verify they are the owner of the Google account.

Google-DynamicAccountCreation Tree With Create Password Node



Node name: Create Password

minPasswordLength: 8

Data Store Decision Node

The Data Store Decision authentication node verifies that the username and password values exist in the data store configured in the realm.

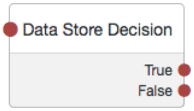
For example, the username and password could be obtained by a combination of the Username Collector and Password Collector nodes, or the Zero Page Login Collector node.

The tree evaluation continues along the `True` path if the credentials are located in the configured data store and the user account profile is not locked. Otherwise, the tree evaluation continues along the `False` path.

Note

Unlike the "LDAP Decision Node", which supports LDAP Behera Password Policies, the data store decision node does not have separate outcomes for accounts that are locked or their password has expired; both result in the `False` path.

The node will also not lockout an account by itself. Instead use a combination of the "Retry Limit Decision Node" and the "Account lockout Node" to lockout an account after repeated failed attempts to authenticate.



Properties:

This node has no configurable properties.

Failure URL Node

The Failure URL authentication node sets the URL to be redirected to when authentication fails.

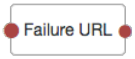
Note

Specifying a failure URL in a tree overrides any `gotoOnFail` query string parameters.

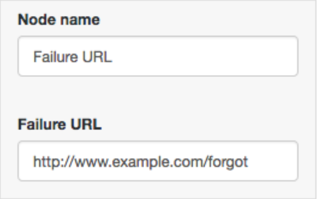
For more information on how AM determines the redirection URL, and to configure the Validation Service to trust redirection URLs, see "Configuring Success and Failure Redirection URLs".

Tip

The URL is also saved into the `sharedState` object, under a property named `failureUrl`, which can be useful for custom node developers. For more information, see "Customizing Authentication Trees".



Properties:



Node name

Failure URL

Failure URL

http://www.example.com/forgot

Property	Usage
Failure URL	Specify the full URL to be redirected to when authentication fails.

Get Session Data Node

The Get Session Data authentication node retrieves the value of a specified key from a user's session data, and stores it in the specified key in the tree's `sharedState` object.

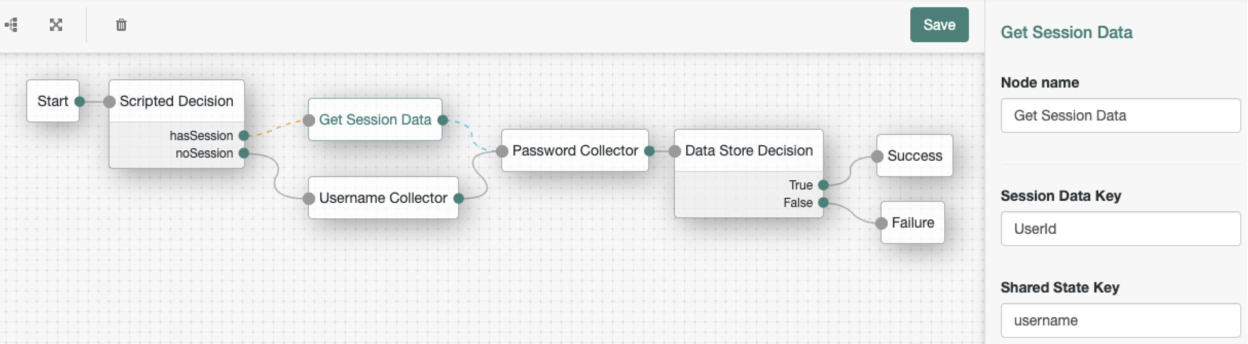
The Get Session Data authentication node is only used during session upgrade —when the user has already successfully authenticated previously— and is now upgrading their session for additional access. For more information on upgrading a session, see "Session Upgrade".

The node will fail with an error if you attempt to get a property when the user does not have an existing session. Use a "Scripted Decision Node" to determine if an existing session is present.

+ *Example Check for Existing Session Script*

```
if (typeof existingSession !== 'undefined')
{
  outcome = "hasSession";
}
else
{
  outcome = "noSession";
}
```

Example:



Get Session Data

Node name: Get Session Data

Session Data Key: UserId

Shared State Key: username

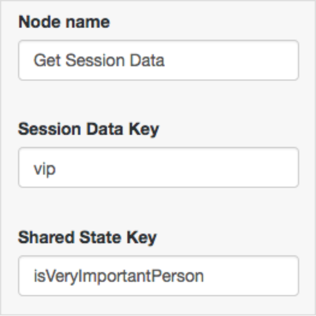
The following table includes example keys that may be available in an existing session, and sample data that they might contain:

Get Session Data Example Keys and Values

Key	Sample value
`AMCtxId`	`e370cca2-02d6-41f9-a244-2b107206bd2a-122934`
`amlbcookie`	`01`
`authInstant`	`2018-04-04T09:19:05Z`

Key	Sample value
`AuthLevel`	`0`
`CharSet`	`UTF-8`
`clientType`	`genericHTML`
`FullLoginURL`	`/openam/UI/Login?realm=%2F`
`Host`	`198.51.100.1`
`HostName`	`openam.example.com`
`Locale`	`en_US`
`Organization`	`dc=openam,dc=forgerock,dc=org`
`Principal`	`id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org`
`Principals`	`amadmin`
`Service`	`ldapService`
`successURL`	`/openam/console`
`sun.am.UniversalIdentifier`	`id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org`
`UserId`	`amadmin`
`UserProfile`	`Required`
`UserToken`	`amadmin`
`webhooks`	`myWebHook`

Properties:



Node name

Get Session Data

Session Data Key

vip

Shared State Key

isVeryImportantPerson

Property	Usage
Session Data Key	Specify the name of a key in the user's session data from which to retrieve the value.
Shared State Key	Specify the name of a key in the `sharedState` object in which to store the retrieved value.

HOTP Generator Node

The HOTP Generator authentication node creates a string of random digits to be used as a hash-based, one-time password (OTP). The default string length is 8 digits.

ForgeRock recommends that you enable OTP encryption for security reasons.

OTP encryption is enabled by default in new AM 6.5.4 installations, as well as in AM existing installations upgraded to 6.5.4. In these installations, AM automatically encrypts and stores OTPs in the shared state configuration. Both the encrypted password and its timestamp are stored in the `oneTimePasswordEncrypted` shared state variable.

Important

- If you're using using an existing AM deployment that has not been upgraded to 6.5.4., you must manually enable OTP encryption. This will impact any existing scripts that use the `oneTimePassword` property in the shared state.

  See the following section "How Do I Configure One-Time-Password Encryption?"

- You can manually disable OTP encryption, although this is not recommended. When not encrypted, the OTP is stored in the `oneTimePassword` variable, and its timestamp is stored as clear text in the `oneTimePasswordTimestamp` variable.

  See the following section "How Do I Configure One-Time-Password Encryption?"

+ *How Do I Configure One-Time-Password Encryption?*

1. Map a secret to the `am.authn.nodes.sharedstate.encryption` secret ID.

   For more information, see "Configuring Secret Stores" in the *Setup and Maintenance Guide*.

2. Set the `org.forgerock.am.auth.node.otp.encrypted` advanced server property to `true`, if needed. New installations of AM 6.5.4 or later already have this property enabled by default.

   You can disable OTP encryption at your own risk. Set the `org.forgerock.am.auth.node.otp.encrypted` advanced server property to `false`.

   + *How Do I Configure Advanced Server Properties?*

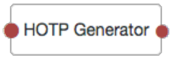
   - To configure advanced server properties for all the instances of the AM environment, in the AM Admin UI, go to Configure > Server Defaults > Advanced.

   - To configure advanced server properties for a particular instance, go to Deployment > Servers > *Server Name* > Advanced.

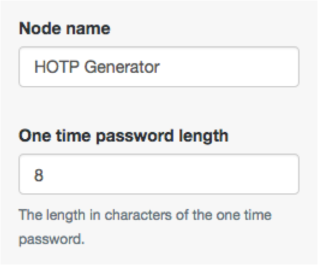
   If the property you want to add or edit is not already configured, add it with its value, then click on the plus (✚) button.

If the property you want to add or edit is already configured, click on the pencil (✎) button to edit it. When you are finished, click on the tick (✔) button.

Save your changes.



Properties:



Node name

HOTP Generator

One time password length

8

The length in characters of the one time password.

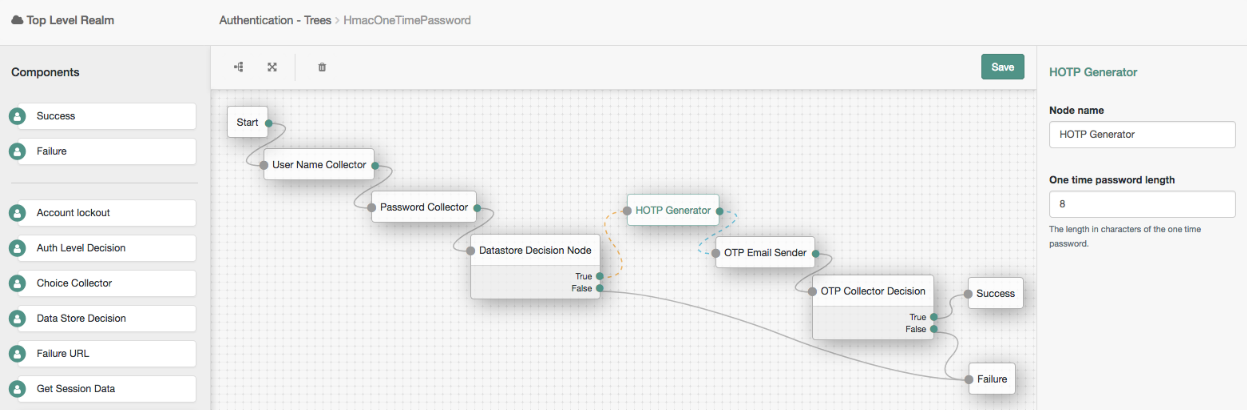
Property	Usage
One-time password length	Specify the number of digits in the one-time password.

Use alongside the following authentication nodes to add one-time password verification to the authentication tree:

- OTP Email Sender Node
- OTP SMS Sender Node
- OTP Collector Decision Node

Example:

HmacOneTimePassword Tree With HOTP Generator Node



HOTP Generator

Node name

HOTP Generator

One time password length

8

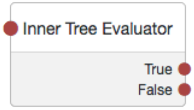
The length in characters of the one time password.

Inner Tree Evaluator Node

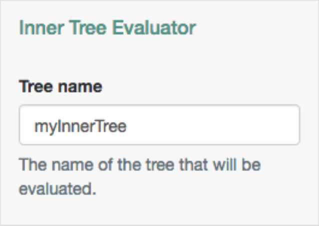
The Inner Tree Evaluator authentication node allows the nesting and evaluation of authentication trees as children within a parent tree. There is no limit to the depth of nested trees.

Any information collected or set by the parent tree, for example, a username or the authentication level, is available to the child trees. Information collected by child trees is available to the parent once evaluation of the child is complete.

The tree evaluation continues along the `True` path if the child tree reached the Success exit point. Otherwise, the tree evaluation continues along the `False` path.



Properties:



Inner Tree Evaluator

Tree name

myInnerTree

The name of the tree that will be evaluated.

Property	Usage
Tree name	Enter the name of the tree to evaluate.

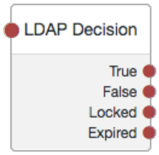
LDAP Decision Node

The LDAP Decision authentication node verifies that the provided username and password values exist in a specified LDAP user data store, and whether they are expired or locked out.

For example, the username and password could be obtained by a combination of the Username Collector and Password Collector nodes, or by using the Zero Page Login Collector node.

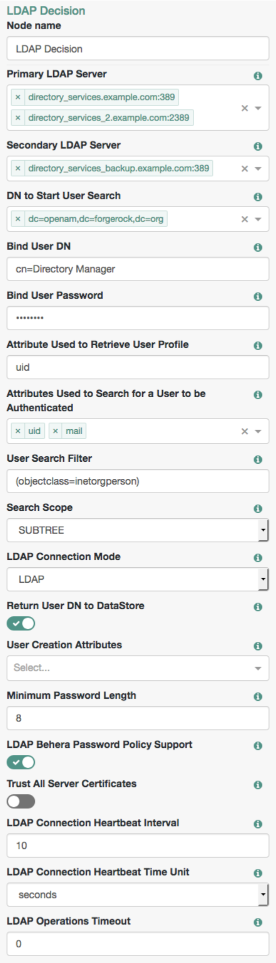
The tree evaluation continues along the `True` outcome path if the credentials are located in the specified LDAP user data store. If the profile associated with the username and password is locked, or the password has expired, tree evaluation continues along the respective `Locked` or `Expired` outcome paths.

If the credentials are not found, the tree evaluation continues along the `False` outcome path.



> **Important**
>
> The LDAP Decision node *requires* specific user attributes in the LDAP user data store. These required attributes are present by default in ForgeRock Directory Services. If you are using an alternative identity store, you might need to modify your LDAP schema to use this node.

Properties:



LDAP Decision

Node name: LDAP Decision

Primary LDAP Server: directory_services.example.com:389, directory_services_2.example.com:2389

Secondary LDAP Server: directory_services_backup.example.com:389

DN to Start User Search: dc=openam,dc=forgerock,dc=org

Bind User DN: cn=Directory Manager

Bind User Password: ••••••••

Attribute Used to Retrieve User Profile: uid

Attributes Used to Search for a User to be Authenticated: uid, mail

User Search Filter: (objectclass=inetorgperson)

Search Scope: SUBTREE

LDAP Connection Mode: LDAP

Return User DN to DataStore: ☑

User Creation Attributes: Select...

Minimum Password Length: 8

LDAP Behera Password Policy Support: ☑

Trust All Server Certificates: ☐

LDAP Connection Heartbeat Interval: 10

LDAP Connection Heartbeat Time Unit: seconds

LDAP Operations Timeout: 0

Property	Usage
Primary LDAP Server	Specify one or more primary directory servers. Specify each directory server in the following format: *host*:*port*. For example, directory_services.example.com:389.
Secondary LDAP Server	Specify one or more secondary directory servers. Specify each directory server in the following format: *host*:*port*. Secondary servers are used when none of the primary servers are available. For example, directory_services_backup.example.com:389.
DN to Start User Search	Specify the DN from which to start the user search. More specific DNs, such as ou=sales,dc=example,dc=com, result in better search performance.

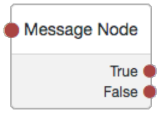
Property	Usage
	If multiple entries exist in the store with identical attribute values, ensure this property is specific enough to return only one entry.
Bind User DN, Bind User Password	Specifies the credentials used to bind to the LDAP user data store.
Attribute Used to Retrieve User Profile	Specifies the attribute used to retrieve the profile of a user from the directory server. The user search will have already happened, as specified by the Attributes Used to Search for a User to be Authenticated and User Search Filter properties.
Attributes Used to Search for a User to be Authenticated	Specifies the attributes used to match an entry in the directory server to the credentials provided by the user. The default value of `uid` will form the following search filter of `uid=user`. Specifying multiple values such as `uid` and `cn` causes the node to create a search filter of `(|(uid=user)(cn=user))`. Multiple attribute values allow the user to authenticate with any one of the values. For example, if you have both `uid` and `mail`, then Barbara Jensen can authenticate with either `bjensen` or `bjensen@example.com`.
User Search Filter	Specifies an additional filter to append to user searches. For example, searching for `mail` and specifying a User Search Filter of `(objectClass=inetOrgPerson)`, causes AM to use `(&(mail=`*`address`*`)(objectClass=inetOrgPerson))` as the resulting search filter, where *address* is the mail address provided by the user.
Search Scope	Specifies the extent of searching for users in the directory server. Scope `OBJECT` means search only the entry specified as the DN to Start User Search, whereas `ONELEVEL` means search only the entries that are directly children of that object. `SUBTREE` means search the entry specified and every entry under it. Default: `SUBTREE`
LDAP Connection Mode	Specifies whether to use SSL or StartTLS to connect to the LDAP user data store. AM must be able to trust the certificates used. Possible values: `LDAP`, `LDAPS`, and `StartTLS` Default: LDAP
Return User DN to DataStore	When enabled, the node returns the DN rather than the User ID. From the DN value, AM uses the RDN to search for the user profile. For example, if a returned DN value is `uid=demo,ou=people,dc=openam,dc=example,dc=org`, AM uses `uid=demo` to search the data store. Default: Enabled
User Creation Attributes	This list lets you map (external) attribute names from the LDAP directory server to (internal) attribute names used by AM.

Property	Usage
Minimum Password Length	Specifies the minimum acceptable password length. Default: `8`
LDAP Behera Password Policy Support	When enabled, support interoperability with servers that implement the Internet-Draft, Password Policy for LDAP Directories. Default: Enabled
Trust All Server Certificates	When enabled, blindly trust server certificates, including self-signed test certificates. Default: Disabled
LDAP Connection Heartbeat Interval	Specifies how often AM should send a heartbeat request to the directory server to ensure that the connection does not remain idle. Some network administrators configure firewalls and load balancers to drop connections that are idle for too long. You can turn this off by setting the value to `0` or to a negative number. Set the units for the interval in the LDAP Connection Heartbeat Time Unit property. Default: `10`
LDAP Connection Heartbeat Time Unit	Specifies the time unit corresponding to LDAP Connection Heartbeat Interval. Default: Seconds
LDAP Operations Timeout	Defines the timeout in milliseconds that AM should wait for a response from the directory server. Default: `0` (means no timeout)

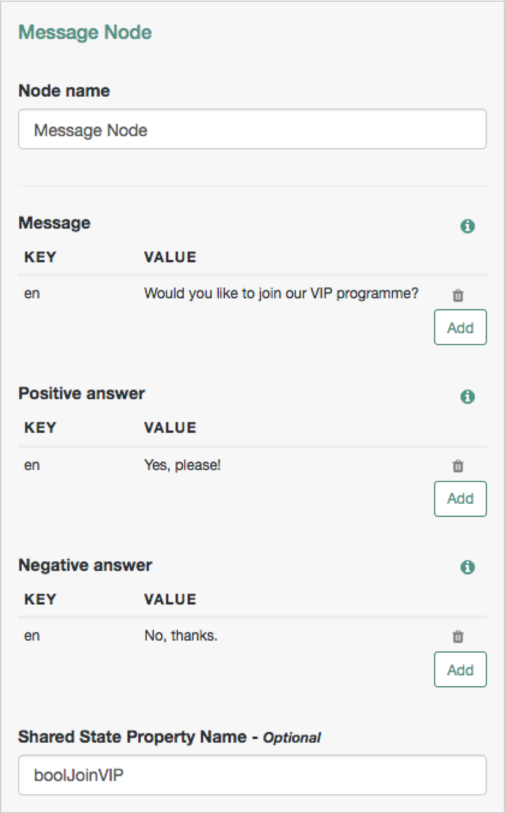
Message Node

The Message authentication node allows you to present a custom, localized message to the user.

In addition to the message, you can provide a localized positive, and negative response that the user must select to proceed.



Properties:



Message Node

Node name

Message Node

Message

KEY	VALUE
en	Would you like to join our VIP programme?

Positive answer

KEY	VALUE
en	Yes, please!

Negative answer

KEY	VALUE
en	No, thanks.

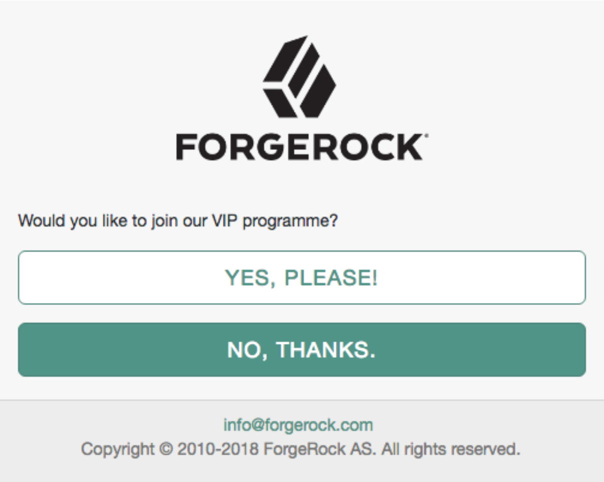
Shared State Property Name - *Optional*

boolJoinVIP

Property	Usage
Message	Click the Add button, and then enter the locale of the message in the `Key` field, and the message to display to the user in the `Value` field. Locales that you specify here must be *real* locales, otherwise AM returns an `Invalid config`> error. If the locale of the user's browser does not match any locale configured in the node, the node uses the Default Authentication Locale (set, per realm, in Authentication > Settings > General). If there is no default authentication locale, the node uses the Default Locale (set in Deployment > Servers > *Server Name* >General > System). If the message property is left blank, the text `Default message` is displayed to the user. To remove a message, select its Delete icon (🗑).
Positive answer	Specify a positive answer that will cause tree evaluation to continue along the `True` outcome path.

Property	Usage
	Click the Add button, and then enter the locale of the positive answer in the `Key` field, and the message to display to the user in the `Value` field. If the locale of the user's browser cannot be determined during authentication, the first message in the list is used. If the message property is left blank, the text `Yes` is displayed to the user. To remove a message, select its Delete icon (🗑).
Negative answer	Specify a negative answer that will cause tree evaluation to continue along the `False` outcome path. Click the Add button, and then enter the locale of the negative answer in the `Key` field, and the message to display to the user in the `Value` field. If the locale of the user's browser cannot be determined during authentication, the first message in the list is used. If the message property is left blank, the text `No` is displayed to the user. To remove a message, select its Delete icon (🗑).

Example:



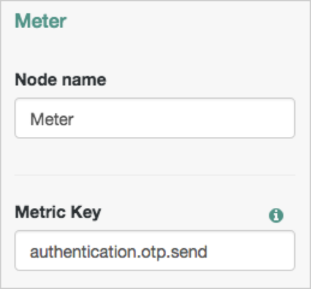
Would you like to join our VIP programme?

info@forgerock.com
Copyright © 2010-2018 ForgeRock AS. All rights reserved.

Meter Node

The Meter authentication node increments a specified metric key each time tree evaluation passes through the node. For information on the `Meter` metric type, see "Monitoring Metric Types" in the *Setup and Maintenance Guide*. The metric is exposed in all available interfaces, as described in "Monitoring Interfaces" in the *Setup and Maintenance Guide*.

Properties:



Meter

Node name

Meter

Metric Key

authentication.otp.send

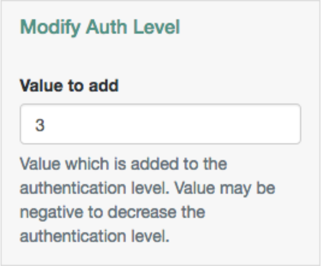
Property	Usage
Metric Key	Specify the name of a metric to increment when tree evaluation passes through the node.

Modify Auth Level Node

The Modify Auth Level authentication node lets you increase or decrease the current authentication level value.

The tree evaluation continues along the single outcome path after modifying the authentication level.

Properties:



Modify Auth Level

Value to add

3

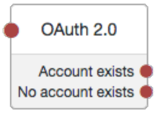
Value which is added to the authentication level. Value may be negative to decrease the authentication level.

Property	Usage
Value to add	Enter a positive integer to increase the current authentication level, or a negative integer to decrease the current authentication level by the specified value.

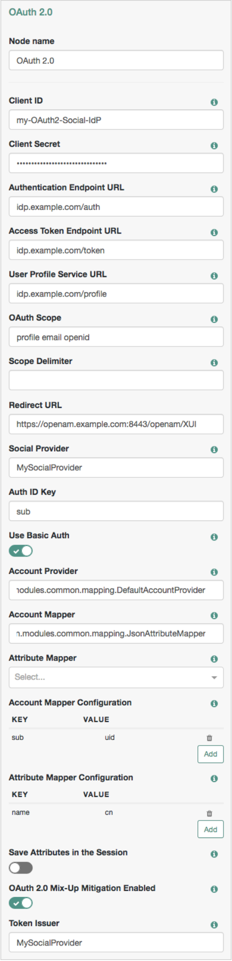
OAuth 2.0 Node

The OAuth 2.0 authentication node lets AM authenticate users of OAuth 2.0-compliant resource servers. References in this section are to RFC 6749, The OAuth 2.0 Authorization Framework.

The tree evaluation continues along the `Account Exists` path if an account matching the attributes retrieved from the social identity provider is found in the user data store. Otherwise, the tree evaluation continues along the `No account exists` path.



Properties:



Node name: OAuth 2.0

Client ID: my-OAuth2-Social-IdP

Client Secret: ••••••••••••••••••••••••••••••

Authentication Endpoint URL: idp.example.com/auth

Access Token Endpoint URL: idp.example.com/token

User Profile Service URL: idp.example.com/profile

OAuth Scope: profile email openid

Scope Delimiter:

Redirect URL: https://openam.example.com:8443/openam/XUI

Social Provider: MySocialProvider

Auth ID Key: sub

Use Basic Auth: ✓

Account Provider: ıodules.common.mapping.DefaultAccountProvider

Account Mapper: ı.modules.common.mapping.JsonAttributeMapper

Attribute Mapper: Select...

Account Mapper Configuration

KEY	VALUE
sub	uid

Attribute Mapper Configuration

KEY	VALUE
name	cn

Save Attributes in the Session: ○

OAuth 2.0 Mix-Up Mitigation Enabled: ✓

Token Issuer: MySocialProvider

Property	Usage
Client ID	Specifies the `client_id` parameter as described in section 2.2 of The OAuth 2.0 Authorization Framework (RFC 6749).
Client Secret	Specifies the `client_secret` parameter as described in section 2.3 of The OAuth 2.0 Authorization Framework (RFC 6749).
Authentication Endpoint URL	Specifies the URL to the social provider's endpoint handling authentication as described in section 3.1 of The OAuth 2.0 Authorization Framework (RFC 6749). Example: `https://accounts.google.com/o/oauth2/v2/auth`
Access Token Endpoint URL	Specifies the URL to the endpoint handling access tokens as described in section 3.2 of The OAuth 2.0 Authorization Framework (RFC 6749). Example: `https://www.googleapis.com/oauth2/v4/token`
User Profile Service URL	Specifies the user profile URL that returns profile information. Exaple: `https://www.googleapis.com/oauth2/v3/userinfo`
OAuth Scope	Specifies a list of user profile attributes that the client application requires, according to *The OAuth 2.0 Authorization Framework (RFC 6749)*. Ensure you use the correct scope delimiter as required by the identity provider, for example commas or spaces. The list depends on the permissions that the resource owner, such as the end user, grants to the client application.
Scope Delimiter	Specifies the delimiter used to separate scope values. Some authorization servers use non-standard separators for scopes, for example commas.
Redirect URL	Specifies the URL the user is redirected to by the social identity provider after authenticating. For authentication trees in AM, set this property to the URL of the XUI, for example `https://openam.example.com:8443/openam/XUI/`.
Social Provider	Specifies the name of the social provider for which this module is being set up. Example: `Google`
Auth ID Key	Specifies the attribute the social identity provider uses to identify an authenticated individual. Example: `id`
Use Basic Auth	Specifies that the client uses HTTP Basic authentication when authenticating to the social provider. Default: `true`
Account Provider	Specifies the name of the class that implements the account provider. Default: `org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider`

Property	Usage

Account Mapper	Specifies the name of the class that implements the method of locating local accounts based on the attributes returned from the social identity provider.
	Provided implementations are:
	`org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper`
	The Account Mapper classes can take two constructor parameters: a comma-separated list of attributes and a prefix to apply to their values. For example, to prefix all received property values with `facebook-` before searching, specify:
	`org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper|*|facebook-`

Attribute Mapper	Specifies the list of fully qualified class names for implementations that map attributes from the OAuth 2.0 authorization server to AM profile attributes.
	Provided implementations are:
	`org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper`
	The Attribute Mapper classes can take two constructor parameters: a comma-separated list of attributes and a prefix to apply to their values, to help differentiate between the providers. For example, to prefix all incoming values with `facebook-`, specify:
	`org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper|*|facebook-`
	Be aware however using an asterisk applies the prefix to all values, including email addresses, postal addresses, and so on.

Account Mapper Configuration	Specifies the attribute configuration used to map the account of the user authenticated in the OAuth 2.0 provider to the local data store in AM. Valid values are in the form *`provider-attr=local-attr`*.
	Examples: `email=mail` and `id=facebook-id`.
	Tip
	When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings, by using dot notation.
	For example, given a JSON payload of:
	`{`
	`  "sub" : "12345",`
	`  "name" : {`
	`    "first_name" : "Demo",`
	`    "last_name" : "User"`
	`  }`
	`}`

Property	Usage
	You can create a mapper such as: `name.first_name=cn`

Attribute Mapper Configuration	Map of OAuth 2.0 provider user account attributes to local user profile attributes, with values in the form *`provider-attr=local-attr`*.
	Examples: `first_name=givenname`, `last_name=sn`, `name=cn`, `email=mail`, `id=facebook-id`, `first_name=facebook-fname`, `last_name=facebook-lname`, `email=facebook-email`.
	Tip
	When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings, by using dot notation.
	For example, given a JSON payload of:
	`{`
	`  "sub" : "12345",`
	`  "name" : {`
	`    "first_name" : "Demo",`
	`    "last_name" : "User"`
	`  }`
	`}`
	You can create a mapper such as:
	`name.first_name=cn`

Save attributes in the session	When enabled, saves the attributes in the Attribute Mapper Configuration field to the AM session.
OAuth 2.0 Mix-Up Mitigation Enabled	Controls whether the OAuth 2.0 authentication node carries out additional verification steps when it receives the authorization code from the authorization server. Specifies that the client must compare the issuer identifier of the authorization server upon registration with the issuer value returned in the `iss` response parameter. If they do not match, the client must abort the authorization process. The client must also confirm that the authorization server's response is intended for the client by comparing the client's client identifier to the value of the `client_id` response parameter. The Token Issuer property must be entered when the OAuth 2.0 Mix-Up Mitigation feature is enabled, so that the validation can succeed. The authorization code response will contain an issuer value (`iss`) that will be validated by the client.

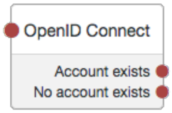
Property	Usage
	Note Consult with the authorization server's documentation on what value it uses for the issuer field. For more information, see section 4 of OAuth 2.0 Mix-Up Mitigation Draft.
Token Issuer	Corresponds to the expected issuer identifier value in the `iss` field of the ID token. Example: `https://accounts.google.com`

OpenID Connect Node

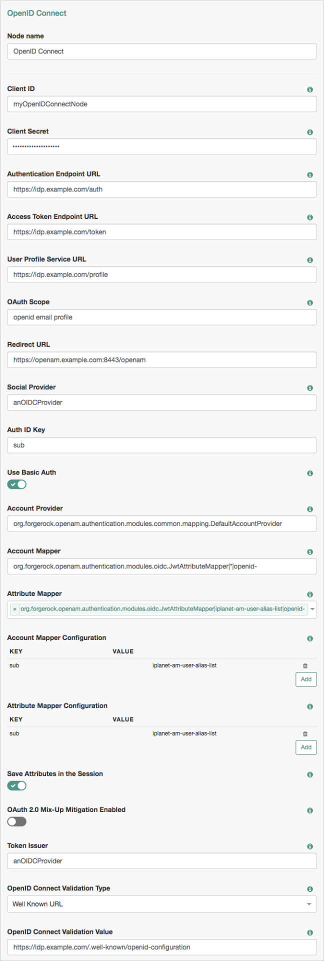
The OpenID Connect authentication node lets AM authenticate users of OpenID Connect-compliant resource servers. As OpenID Connect is an additional layer on top of OAuth 2.0, many references in this section are to RFC 6749, The OAuth 2.0 Authorization Framework. OpenID Connect-specific references are to the OpenID Connect Core 1.0 incorporating errata set 1 specification.

The tree evaluation continues along the `Account Exists` path if an account matching the attributes retrieved from the OpenID Connect identity provider is found in the identity store. Otherwise, the tree evaluation continues along the `No account exists` path.

The OpenID Connect node implements the "Authorization Code Grant" flow.



Properties:



OpenID Connect

Node name: OpenID Connect

Client ID: myOpenIDConnectNode

Client Secret: ••••••••••••••••••••

Authentication Endpoint URL: https://idp.example.com/auth

Access Token Endpoint URL: https://idp.example.com/token

User Profile Service URL: https://idp.example.com/profile

OAuth Scope: openid email profile

Redirect URL: https://openam.example.com:8443/openam

Social Provider: anOIDCProvider

Auth ID Key: sub

Use Basic Auth: ☑

Account Provider: org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider

Account Mapper: org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper|*|openid-

Attribute Mapper: org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper|iplanet-am-user-alias-list|openid-

Account Mapper Configuration

KEY	VALUE
sub	iplanet-am-user-alias-list

Attribute Mapper Configuration

KEY	VALUE
sub	iplanet-am-user-alias-list

Save Attributes in the Session: ☑

OAuth 2.0 Mix-Up Mitigation Enabled: ☐

Token Issuer: anOIDCProvider

OpenID Connect Validation Type: Well Known URL

OpenID Connect Validation Value: https://idp.example.com/.well-known/openid-configuration

Property	Usage
Client ID	Specifies the `client_id` parameter as described in section 2.2 of The OAuth 2.0 Authorization Framework (RFC 6749).
Client Secret	Specifies the `client_secret` parameter as described in section 2.3 of The OAuth 2.0 Authorization Framework (RFC 6749).
Authentication Endpoint URL	Specifies the URL to the social provider's endpoint handling authentication as described in section 3.1 of The OAuth 2.0 Authorization Framework (RFC 6749). Example: `https://accounts.google.com/o/oauth2/v2/auth`
Access Token Endpoint URL	Specifies the URL to the endpoint handling access tokens as described in section 3.2 of The OAuth 2.0 Authorization Framework (RFC 6749).

Property	Usage
	Example: `https://www.googleapis.com/oauth2/v4/token`
User Profile Service URL	Specifies the user profile URL that returns profile information. If not specified, attributes are mapped from the claims returned by the `id_token`, and no call to a user profile endpoint is made. Exaple: `https://www.googleapis.com/oauth2/v3/userinfo`
OAuth Scope	Specifies a list of user profile attributes that the client application requires, according to *The OAuth 2.0 Authorization Framework (RFC 6749)* . Ensure you use the correct scope delimiter as required by the identity provider, for example commas or spaces. The list depends on the permissions that the resource owner, such as the end user, grants to the client application.
Redirect URL	Specifies the URL the user is redirected to by the social identity provider after authenticating. For authentication trees in AM, set this property to the URL of the XUI, for example `https://openam.example.com:8443/openam/XUI/`.
Social Provider	Specifies the name of the OpenID Connect provider for which this node is being set up. Example: `Google`
Auth ID Key	Specifies the attribute the social identity provider uses to identify an authenticated individual. Example: `sub`
Use Basic Auth	Specifies that the client uses HTTP Basic authentication when authenticating to the social provider. Default: `true`
Account Provider	Specifies the name of the class that implements the account provider. Default: `org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider`
Account Mapper	Specifies the name of the class that implements the method of locating local accounts based on the attributes returned from the social identity provider. The provided implementations is `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper`. The Account Mapper classes can take two constructor parameters: a comma-separated list of attributes and a prefix to apply to their values. For example, to prefix all received property values with `openid-` before searching, specify: `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper|*|openid-`

Property	Usage
Attribute Mapper	Specifies the list of fully qualified class names for implementations that map attributes from the authorization server to AM profile attributes. The provided implementations is `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper`. The Attribute Mapper classes can take two constructor parameters: a comma-separated list of attributes and a prefix to apply to their values, to help differentiate between the providers. For example, to prefix incoming `iplanet-am-user-alias-list` values with `openid-`, specify: `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper|iplanet-am-user-alias-list|openid-` To prefix all incoming values use an asterisk (`*`) as the attribute list. Note that an asterisk prefixes all values, including email addresses, postal addresses, and so on.
Account Mapper Configuration	Specifies the attribute configuration used to map the account of the user authenticated in the provider to the local identity store in AM. To add a mapping, specify the name of the provider attribute as the Key, and the local attribute to map to as the Value. For example, click the Add button, then specify `sub` in the Key field and `iplanet-am-user-alias-list` in the Value field, and then click the Plus button (✚).

Attribute Mapper Configuration	Specifies how to map provider user attributes to local user profile attributes.
	To add a mapping, specify the name of the provider attribute as the Key, and the local attribute to map to as the Value.
	For example, click the Add button, then specify `id` in the Key field and `facebook-id` in the Value field, and then click the Plus button (✚).
	Examples:
	`first_name=givenname`
	`last_name=sn`
	`name=cn`
	`email=mail`
	`id=facebook-id`
	`first_name=facebook-fname`
	`last_name=facebook-lname`
	`email=facebook-email`

Save attributes in the session	When enabled, saves the attributes in the Attribute Mapper Configuration field to the AM session.
OAuth 2.0 Mix-Up Mitigation Enabled	Controls whether the authentication node carries out additional verification steps when it receives the authorization code from the authorization server. Specifies that the client must compare the issuer identifier of the authorization server upon registration with the issuer value returned in the `iss` response

Property	Usage
	parameter. If they do not match, the client must abort the authorization process. The client must also confirm that the authorization server's response is intended for the client by comparing the client's client identifier to the value of the `client_id` response parameter. The Token Issuer property must be entered when the OAuth 2.0 Mix-Up Mitigation feature is enabled, so that the validation can succeed. The authorization code response will contain an issuer value (`iss`) that will be validated by the client. **Note** Consult with the authorization server's documentation on what value it uses for the issuer field. For more information, see section 4 of OAuth 2.0 Mix-Up Mitigation Draft.
Token Issuer	Corresponds to the expected issuer identifier value in the `iss` field of the ID token. Example: `https://accounts.google.com`

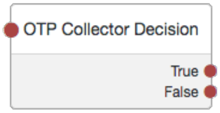
OpenID Connect Validation Type	Specifies how to validate the ID token received from the OpenID Connect provider.
	The following options are available to validate an incoming OpenID Connect ID token:
	`Well Known URL` **(Default)**
	Retrieves the provider's keys based on the information provided in its OpenID Connect configuration URL.
	Specify the provider's configuration URL in the OpenID Connect Validation Value field, for example `https://accounts.google.com/.well-known/openid-configuration`.
	`Client Secret`
	Validates the ID token signature with a specified client secret key.
	Specify the key to use in the OpenID Connect Validation Value field.
	`JWK URL`
	Retrieve the neccessary JSON web key from the URL that you specify.
	Specify the provider's JWK URI in the OpenID Connect Validation Value field, for example `https://www.googleapis.com/oauth2/v3/certs`.
	Note
	AM does not support non-string JWT header parameters, such as `jku` and `jwe`:

Property	Usage
	- AM 6.5.2 and later parse the JWT but ignores the non-string header parameters. - AM versions earlier than 6.5.2 do not parse the JWT, and will return an HTTP 500 error message. Configure the public keys/certificates in AM instead of adding the headers, as explained in the relevant sections of the documentation.
OpenID Connect Validation Value	Provide the URL or secret key used to verify an incoming ID token, depending on the value selected in the OpenID Connect Validation Type property.

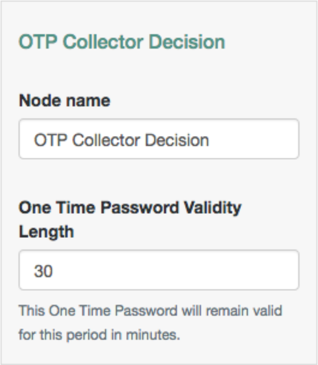
OTP Collector Decision Node

The OTP Collector Decision authentication node requests and verifies one-time passwords.

The tree evaluation continues along the `True` outcome path if the entered one-time password is valid for the authentication in progress. Otherwise, the tree evaluation continues along the `False` outcome path.



Properties:



OTP Collector Decision

Node name

OTP Collector Decision

One Time Password Validity Length

30

This One Time Password will remain valid for this period in minutes.

Property	Usage
One Time Password Validity Length	Specify the length of time, in minutes, that a one-time password remains valid. Default: `5`

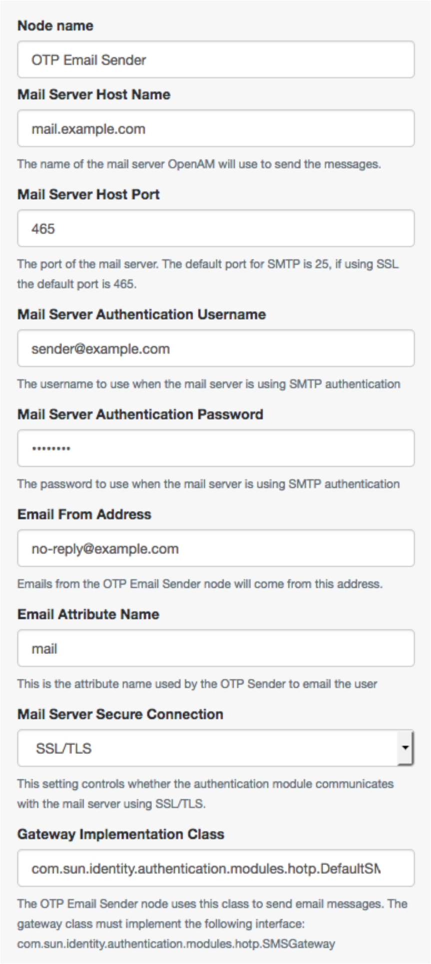
OTP Email Sender Node

The OTP Email Sender authentication node sends an email containing a generated one-time password to the user.

Send mail requests will timeout after 10 seconds.

> **Tip**
>
> You can change the timeout in the following advanced server properties:
>
> - `org.forgerock.openam.smtp.system.connect.timeout`
> - `org.forgerock.openam.smtp.system.socket.read.timeout`
> - `org.forgerock.openam.smtp.system.socket.write.timeout`
>
> \+ *How Do I Configure Advanced Server Properties?*
>
> - To configure advanced server properties for all the instances of the AM environment, in the AM Admin UI, go to Configure > Server Defaults > Advanced.
> - To configure advanced server properties for a particular instance, go to Deployment > Servers > *Server Name* > Advanced.
>
> If the property you want to add or edit is not already configured, add it with its value, then click on the plus (✚) button.
>
> If the property you want to add or edit is already configured, click on the pencil (✎) button to edit it. When you are finished, click on the tick (✓) button.
>
> Save your changes.
>
> For more information, see Advanced Properties in the *Reference*.

Properties:



Node name

OTP Email Sender

Mail Server Host Name

mail.example.com

The name of the mail server OpenAM will use to send the messages.

Mail Server Host Port

465

The port of the mail server. The default port for SMTP is 25, if using SSL the default port is 465.

Mail Server Authentication Username

sender@example.com

The username to use when the mail server is using SMTP authentication

Mail Server Authentication Password

••••••••

The password to use when the mail server is using SMTP authentication

Email From Address

no-reply@example.com

Emails from the OTP Email Sender node will come from this address.

Email Attribute Name

mail

This is the attribute name used by the OTP Sender to email the user

Mail Server Secure Connection

SSL/TLS

This setting controls whether the authentication module communicates with the mail server using SSL/TLS.

Gateway Implementation Class

com.sun.identity.authentication.modules.hotp.DefaultSM

The OTP Email Sender node uses this class to send email messages. The gateway class must implement the following interface: com.sun.identity.authentication.modules.hotp.SMSGateway

Property	Usage
Mail Server Host Name	Specifies the hostname of the SMTP email server.
Mail Server Host Port	Specifies the outgoing mail server port. Common ports are 25, 465 (when connecting over SSL), or 587 (for StartTLS).
Mail Server Authentication Username	Specifies the username AM uses to connect to the mail server.
Mail Server Authentication Password	Specifies the password AM uses to connect to the mail server.
Email From Address	Specifies the email address from which the one-time password will appear to have been sent.

Property	Usage
Email Attribute Name	Specifies the user's profile attribute containing the email address to which to email the OTP. Default: `mail`
Mail Server Secure Connection	Specifies how to connect to the mail server. If a secure method is specified, AM must trust the server certificate of the mail server. The possible values for this property are: • `NON SSL/TLS` • `SSL/TLS` • `Start TLS` Default: `SSL/TLS`
Gateway Implementation Class	Specifies the class the node uses to send SMS and email messages. A custom class must implement the `com.sun.identity.authentication.modules.hotp.SMSGateway` interface. Default: `com.sun.identity.authentication.modules.hotp.DefaultSMSGatewayImpl`

OTP SMS Sender Node

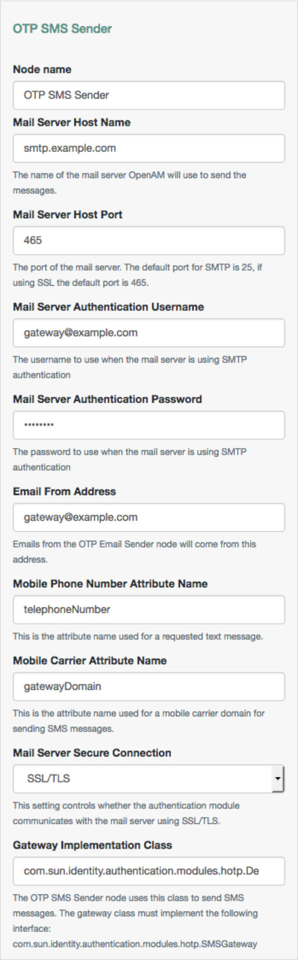
The OTP SMS Sender authentication node uses an email-to-SMS gateway provider to send an SMS message containing a generated one-time password to the user.

The node sends an email to an address formed by joining the following values together:

- The user's telephone number, obtained by querying a specified profile attribute, for example `telephoneNumber`.
- The @ character.
- The email-to-SMS gateway domain, obtained by querying the profile attribute specified by the Mobile Carrier Attribute Name property.

For example, if configured to use the *TextMagic* email-to-SMS service, the node might send an email through the specified SMTP server to the address: `18005550187@textmagic.com`.

Properties:



OTP SMS Sender

Node name

OTP SMS Sender

Mail Server Host Name

smtp.example.com

The name of the mail server OpenAM will use to send the messages.

Mail Server Host Port

465

The port of the mail server. The default port for SMTP is 25, if using SSL the default port is 465.

Mail Server Authentication Username

gateway@example.com

The username to use when the mail server is using SMTP authentication

Mail Server Authentication Password

••••••••

The password to use when the mail server is using SMTP authentication

Email From Address

gateway@example.com

Emails from the OTP Email Sender node will come from this address.

Mobile Phone Number Attribute Name

telephoneNumber

This is the attribute name used for a requested text message.

Mobile Carrier Attribute Name

gatewayDomain

This is the attribute name used for a mobile carrier domain for sending SMS messages.

Mail Server Secure Connection

SSL/TLS

This setting controls whether the authentication module communicates with the mail server using SSL/TLS.

Gateway Implementation Class

com.sun.identity.authentication.modules.hotp.De

The OTP SMS Sender node uses this class to send SMS messages. The gateway class must implement the following interface: com.sun.identity.authentication.modules.hotp.SMSGateway

Property	Usage
Mail Server Host Name	Specifies the hostname of the SMTP email server.
Mail Server Host Port	Specifies the outgoing mail server port. Common ports are 25, 465 (when connecting over SSL), or 587 (for StartTLS).
Mail Server Authentication Username	Specifies the username AM uses to connect to the mail server.
Mail Server Authentication Password	Specifies the password AM uses to connect to the mail server.
Email From Address	Specifies the email address from which the one-time password will appear to have been sent.

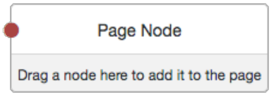
Property	Usage
Mobile Phone Number Attribute Name	Specifies the user's profile attribute containing the mobile phone number to which to send the SMS containing the OTP. Default: `telephoneNumber`
Mobile Carrier Attribute Name	Specifies the user's profile attribute containing the mobile carrier domain used as the email to SMS gateway.
Mail Server Secure Connection	Specifies how to connect to the mail server. If a secure method is specified, AM must trust the server certificate of the mail server. The possible values for this property are: • `NON SSL/TLS` • `SSL/TLS` • `Start TLS` Default: `SSL/TLS`
Gateway Implementation Class	Specifies the class the node uses to send SMS and email messages. A custom class must implement the `com.sun.identity.authentication.modules.hotp.SMSGateway` interface. Default: `com.sun.identity.authentication.modules.hotp.DefaultSMSGatewayImpl`

Page Node

The Page authentication node combines multiple nodes that request input into a single page for display to the user. Drag and drop nodes on to the page node to combine them.

The outcome paths are determined by the last node in the page node. Only the last node in the page can have more than one outcome path.

Only nodes that use callbacks to request input can be added to a page node. Other nodes, such as the Data Store Decision Node and Push Sender Node must not be added to a page node.



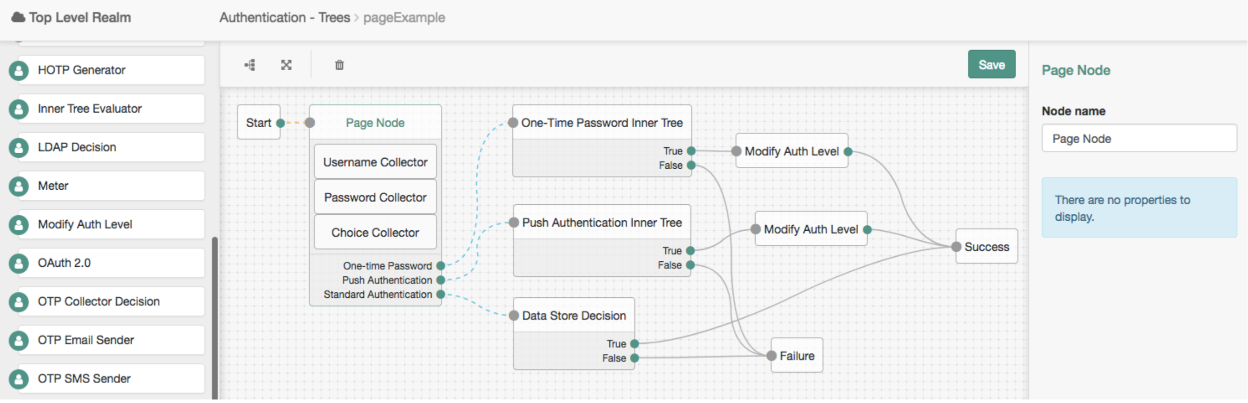
Properties:

This node has no configurable properties.

Example:

The following example uses a page node containing a username collector, a password collector, and a choice collector:

Example Tree With Page Node



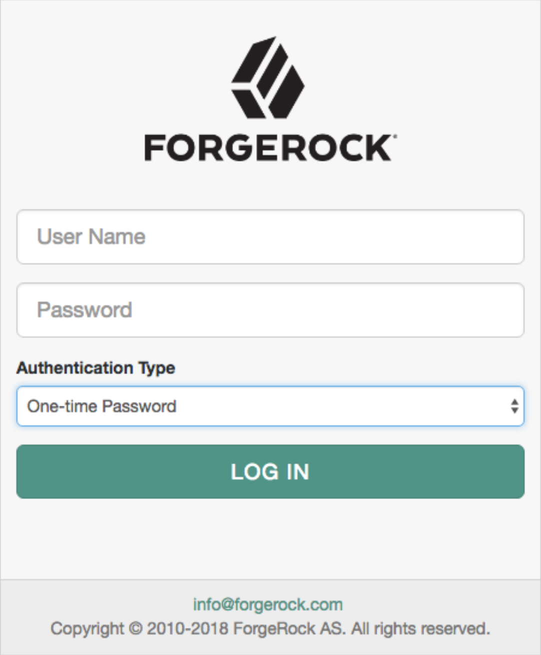
Page Node

Node name: Page Node

There are no properties to display.

The user is presented with all of the requests for input on a single page:

User View of Example Tree with Page Node



Password Collector Node

The Password Collector authentication node prompts the user to enter their password. The captured password is transient, persisting only until the authentication flow reaches the next node requiring user interaction.

The tree evaluation continues along the single outcome path after capturing the password.

Properties:

This node has no configurable properties.

Polling Wait Node

The Polling Wait authentication node pauses progress of the authentication tree for a specified number of seconds, for example in order to wait for a response to a one-time password email or push notification.

Requests to the tree made during the wait period are sent a `PollingWaitCallback` callback and an authentication ID. For example, the following callback indicates a wait time of 10 seconds:

```
{
    "authId": "eyJ0eXAiOiJK...u4WvZmiI",
    "callbacks": [
        {
            "type": "PollingWaitCallback",
            "output": [
                {
                    "name": "waitTime",
                    "value": "10000"
                },
                {
                    "name": "message",
                    "value": "Waiting for response..."
                }
            ]
        }
    ]
}
```

The client must wait 10 seconds before returning the callback data, including the `authId`. For example:

```
$ curl \
--request POST \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--header "Content-Type: application/json" \
--data '{
  "authId": "eyJ0eXAiOiJK...u4WvZmiI",
  "callbacks": [
      {
          "type": "PollingWaitCallback",
          "output": [
              {
                  "name": "waitTime",
                  "value": "10000"
              },
              {
                  "name": "message",
                  "value": "Waiting for response..."
              }
          ]
      }
  ]
}' 'https://openam.example.com:8443/openam/json/realms/root/authenticate?
authIndexType=service&authIndexValue=Example'
#authenticate-indextype

#authenticate-xml
curl -get \
--cookie "iPlanetDirectoryPro=AQIC5w...NTcy*" \" \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--data-urlencode 'authIndexType=composite_advice' \
--data-urlencode 'authIndexValue=<Advices>
    <AttributeValuePair>
        <Attribute name="TransactionConditionAdvice"/>
        <Value>9dae2c80-fe7a-4a36-b57b-4fb1271b0687</Value>
    </AttributeValuePair>
</Advices>' \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
#authenticate-xml

cat << EOF
#authenticate-xml-expected
{
    "authId": "eyJ0eXAiOi...WLxJ-1d6ovYKHQ",
    "template": "",
    "stage": "AuthenticatorPush3",
    "header": "Authenticator Push",
    "callbacks": [
        {
            "type": "PollingWaitCallback",
            "output": [
                {
                    "name": "waitTime",
                    "value": "10000"
                }
            ]
        },
        {
            "type": "ConfirmationCallback",
            "output": [
                {
                    "name": "prompt",
                    "value": ""
                },
                {
                    "name": "messageType",
                    "value": 0
                },
                {
                    "name": "options",
                    "value": [
                        "Use Emergency Code"
                    ]
                },
                {
                    "name": "optionType",
                    "value": -1
                },
                {
                    "name": "defaultOption",
                    "value": 0
                }
            ],
            "input": [
                {
                    "name": "IDToken2",
                    "value": 100
                }
            ]
        }
    ]
}
#authenticate-xml-expected
EOF


#authenticate-indextype
curl \
--cookie "iPlanetDirectoryPro=AQIC5w...NTcy*" \" \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--data '{
    "authId":"eyJ0eXAiOi...WLxJ-1d6ovYKHQ",
    "template":"",
    "stage":"AuthenticatorPush3",
    "header":"Authenticator Push",
    "callbacks":[
        {
            "type":"PollingWaitCallback",
            "output":[
                {
                    "name":"waitTime",
                    "value":"10000"
                }
            ]
        },
        {
            "type":"ConfirmationCallback",
            "output":[
                {
                    "name":"prompt",
                    "value":""
                },
                {
                    "name":"messageType",
                    "value":0
                },
                {
                    "name":"options",
                    "value":[
                        "Use Emergency Code"
                    ]
                },
                {
                    "name":"optionType",
                    "value":-1
                },
                {
                    "name":"defaultOption",
                    "value":0
                }
            ],
            "input":[
                {
                    "name":"IDToken2",
                    "value":100
                }
            ]
        }
    ]
}' \
"https://openam.example.com:8443/openam/json/realms/root/authenticate\
?authIndexType=composite_advice\
&authIndexValue=%3CAdvices%3E%0A\
%3CAttributeValuePair%3E%0A%3CAttribute%20name%3D\
%22TransactionConditionAdvice%22%2F%3E%0A\
%3CValue%3E9dae2c80-fe7a-4a36-b57b-4fb1271b0687\
%3C%2FValue%3E%0A%3C%2FAttributeValuePair\
%3E%0A%3C%2FAdvices%3E"
```

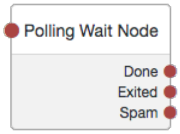
For more information on authenticating using the REST API, see "Authentication and Logout using REST".

When using the XUI for authentication, it automatically waits for the required amount of time and resubmit the page in order to continue tree evaluation. The message displayed whilst waiting is configurable by using the Waiting Message property.

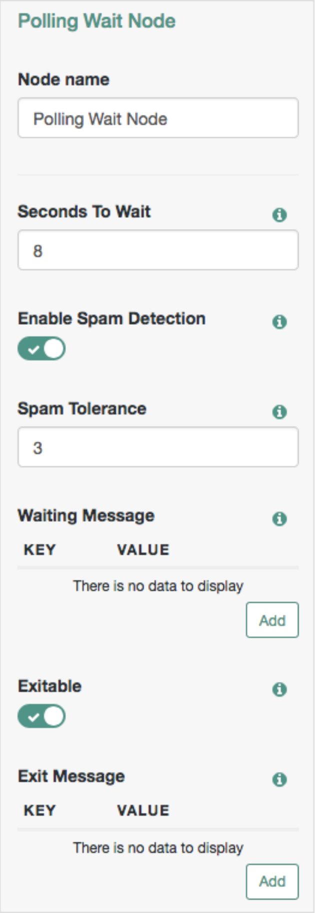
Tree evaluation continues along the `Done` outcome path when the next request is received after the wait time has passed.

Enabling Spam detection adds a `Spam` outcome path to the node. Tree evaluation continues along the `Spam` outcome path if more than the specified number of requests are received during the wait time.

Enabling the user to exit without waiting adds an `Exited` outcome path to the node. Tree evaluation continues along the `Exited` outcome path if the user clicks the button that appears when the option is enabled. The message displayed on the exit button is configurable by using the Exit Message property.



Properties:



Polling Wait Node

Node name: Polling Wait Node

Seconds To Wait: 8

Enable Spam Detection: ☑

Spam Tolerance: 3

Waiting Message

KEY	VALUE
There is no data to display	

Exitable: ☑

Exit Message

KEY	VALUE
There is no data to display	

Property	Usage
Seconds To Wait	Specify the number of seconds to pause the authentication tree. Default: `8`
Enable Spam Detection	Specify whether to track the number of responses received during the wait time, and continue tree evaluation along the `Spam` outcome path if the number specified in the Spam Tolerance property is exceeded.

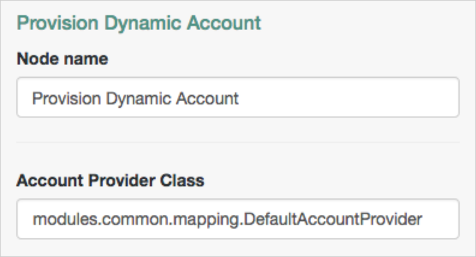
Property	Usage
	Default: Disabled
Spam Tolerance	Specify the number of responses to allow during the wait time before continuing tree evaluation along the `Spam` outcome path. This property only applies if spam detection is enabled. Default: `3`
Waiting Message	Specifies the optional message to display to the user. You can provide the message in multiple languages by specifying the locale in the `KEY` field, for example `en-US`. For information on valid locale strings, see JDK 8 and JRE 8 Supported Locales. The locale selected for display is based on the user's locale settings in their browser. Messages provided in the node override the defaults provided by AM. For information about customizing and translating the default messages, see "*Internationalization*" in the *Authentication Node Development Guide*.
Exitable	Specify whether the user can exit the node during the wait period. Enabling this option adds a button with a configurable message to the page. Clicking the button causes tree evaluation to continue along the `Exited` outcome path. Default: Disabled
Exit Message	Specifies the optional message to display to the user on the button used to exit the node before the wait period has elapsed. For example, `Cancel` or `Lost phone? Use Recovery Code`. This property only applies if the Exitable property is enabled. You can provide the message in multiple languages by specifying the locale in the `KEY` field, for example `en-US`. For information on valid locale strings, see JDK 8 and JRE 8 Supported Locales. The locale selected for display is based on the user's locale settings in their browser. Messages provided in the node override the defaults provided by AM. For information about customizing and translating the default messages, see "*Internationalization*" in the *Authentication Node Development Guide*.

Provision Dynamic Account Node

The Provision Dynamic Account node provisions an account following successful authentication by a social identity provider node. Accounts are provisioned using properties defined in the attribute mapper configuration of a social authentication node earlier in the tree evaluation, for example the OAuth 2.0 Node.

If a password has been acquired from the user, for example by using the Create Password Node, it is used when provisioning the account. Otherwise, a 20 character random string is used.

Properties:



Provision Dynamic Account

Node name: Provision Dynamic Account

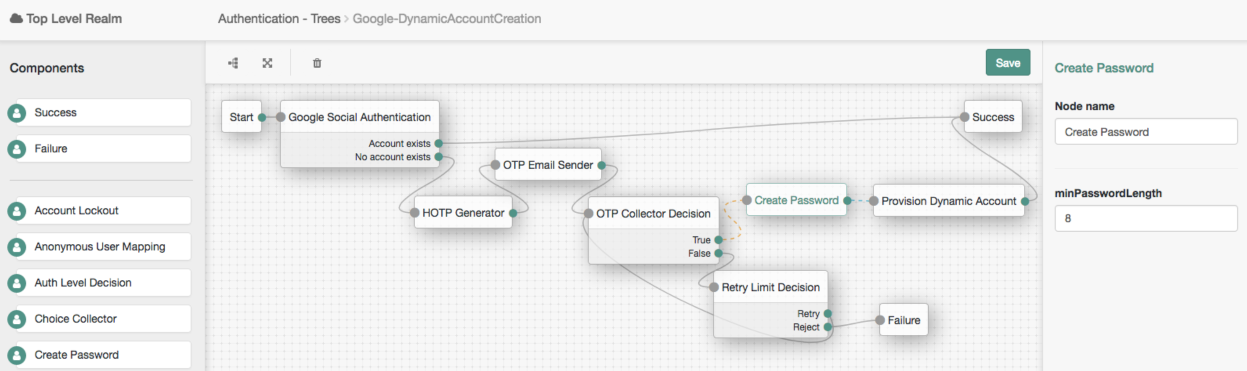
Account Provider Class: modules.common.mapping.DefaultAccountProvider

Property	Usage
Account Provider	Specifies the name of the class that implements the account provider. Default: `org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider`

Example:

The following example uses the Provision Dynamic Account authentication node to allow users who have performed social authentication using Google to provide a password and provision an account, if they do not have a matching existing profile. They must enter a one-time password to verify they are the owner of the Google account.

Google-DynamicAccountCreation Tree With Provision Dynamic Account Node



Create Password

Node name: Create Password

minPasswordLength: 8

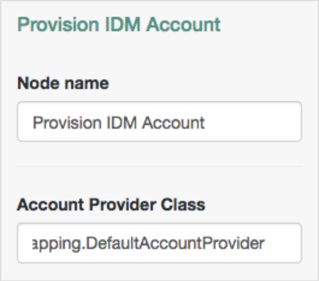
Provision IDM Account Node

The Provision IDM Account node redirects users to an IDM instance to provision an account.

Ensure you have configured the details of the IDM instance in AM, by navigating to Configure > Global Services > IDM Provisioning.

For information on using IDM for provisioning, see Integrating IDM With the ForgeRock Identity Platform in the *Samples Guide*.

Properties:



Provision IDM Account

Node name: Provision IDM Account

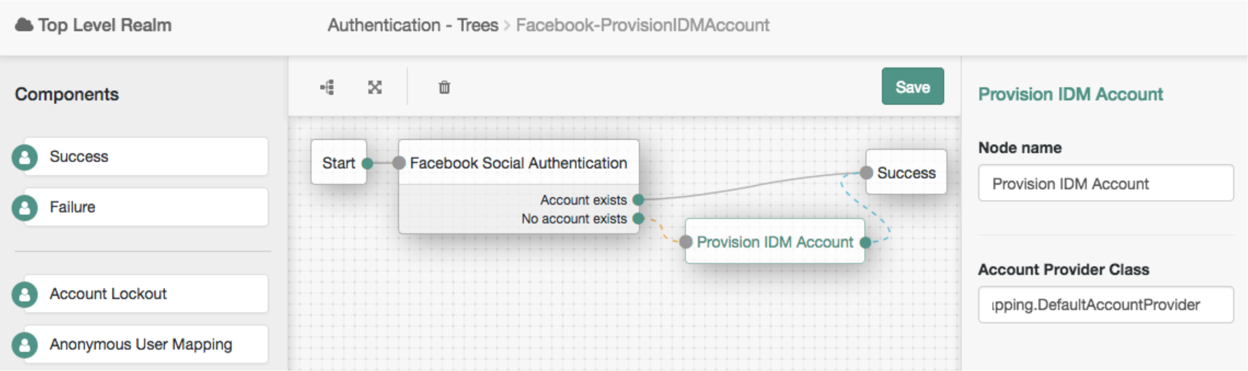
Account Provider Class: apping.DefaultAccountProvider

Property	Usage
Account Provider	Specifies the name of the class that implements the account provider. Default: `org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider`

Example:

The following example uses the Provision IDM Account authentication node to allow users who have performed social authentication using Facebook to provision an account using IDM, if they do not have a matching existing profile.

Facebook-ProvisionIDMAccount Tree With Provision IDM Account Node



Provision IDM Account

Node name: Provision IDM Account

Account Provider Class: ipping.DefaultAccountProvider

Push Result Verifier Node

The Push Result Verifier node works together with the Push Sender Node to validate the user's response to a previously sent push notification message.

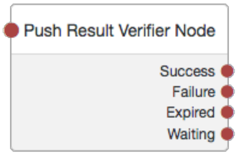
Tree evaluation continues along the `Success` outcome path if the push notification was positively responded to by the user. For example, using the ForgeRock Authenticator app, the user slid the switch with a checkmark on horizontally to the right.

Tree evaluation continues along the `Failure` outcome path if the push notification was negatively responded to by the user. For example, using the ForgeRock Authenticator app, the user tapped the cancel icon in the top-right of the screen.

If the push notification was not responded to within the Message Timeout value specified in the Push Sender Node then tree evaluation continues along the `Expired` outcome path.

If a response to the push message has not yet been received, then tree evaluation continues along the `Waiting` outcome path.

> **Tip**
>
> If the push message contained any additional information, for example if it was a registration request, the values are stored in the `sharedState` object of the tree, in a key named `pushContent`. For information on creating or customizing authentication nodes, see Authentication Node Development Guide.



Properties:

This node has no configurable properties.

Push Sender Node

The Push Sender authentication node sends push notification messages to a device such as a mobile phone, enabling multi-factor authentication.

The Push Sender authentication node requires that the Push Notification Service has also been configured. For information on the properties used by the service, see "Push Notification Service" in the *Reference*. For information on provisioning the credentials used by the service, see How To Configure Service Credentials (Push Auth, Docker) in Backstage in the *ForgeRock Knowledge Base*.

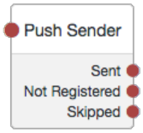
Tree evaluation continues along the `Sent` outcome path if the push notification was successfully sent to the handling service.

If the user does not have a registered device, tree evaluation continues along the `Not Registered` outcome path. To determine whether the user has a registered device, the tree must have already acquired a username, for example by using a Username Collector Node.

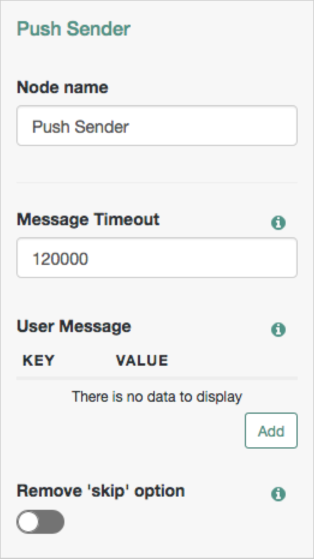
Note

Authentication trees are not capable of registering a device to a profile. For information on using authentication chains to register devices, see "Creating Authentication Chains for Push Authentication".

If the user chooses to skip push authentication, tree evaluation continues along the `Skipped` outcome path. You can configure whether the user is able to skip the node by setting the Two Factor Authentication Mandatory property. See "Letting Users Opt Out of One-Time Password Authentication".



Properties:



Push Sender

Node name

Push Sender

Message Timeout

120000

User Message

KEY	VALUE

There is no data to display

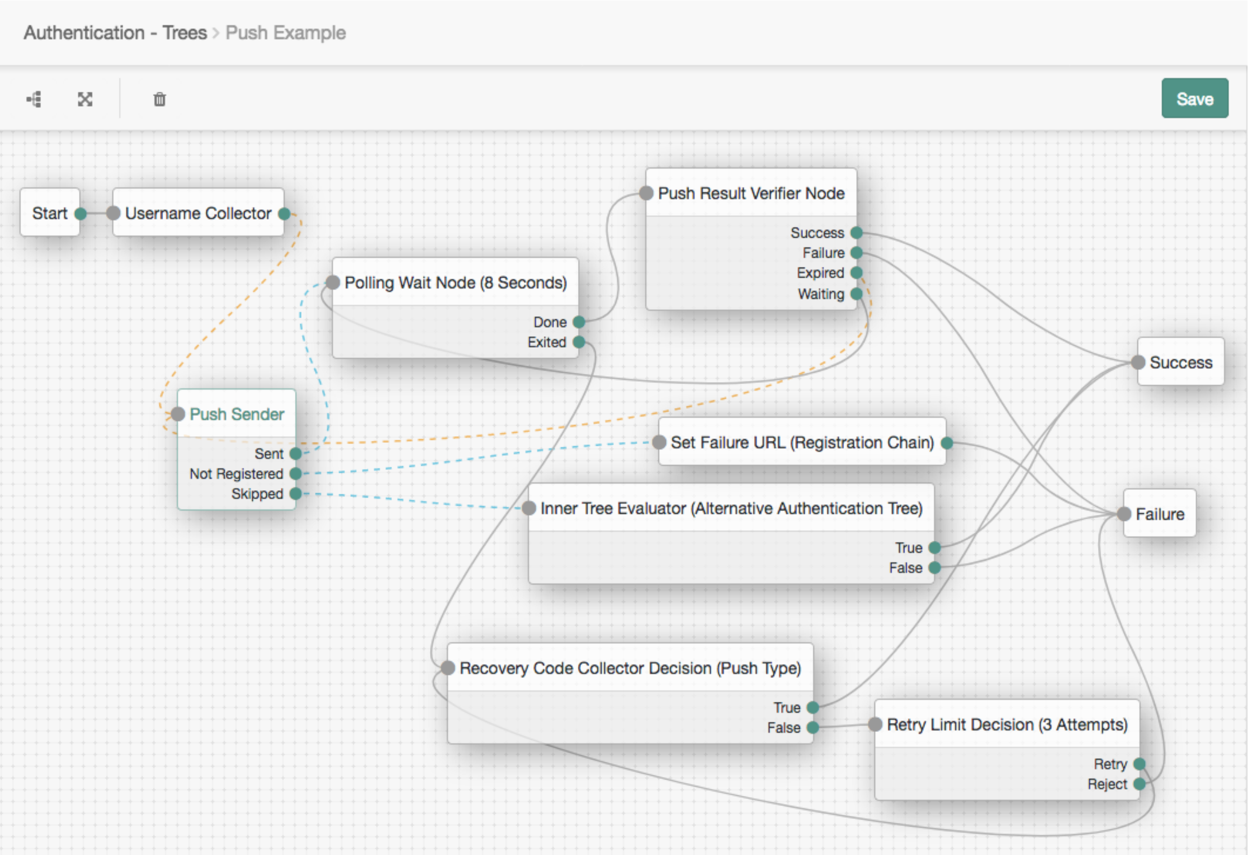
Remove 'skip' option

Property	Usage
Message Timeout	Specifies the number of milliseconds the push notification message will remain valid. The Push Result Verifier Node rejects responses to push messages that have timed out.
User Message	Specifies the optional message to send to the user. You can provide the message in multiple languages by specifying the locale in the `KEY` field, for example `en-US`. For information on valid locale strings, see JDK 11 Supported Locales. The locale selected for display is based on the user's locale settings in their browser.

Property	Usage
	Messages provided in the node override the defaults provided by AM. For information about customizing and translating the default messages, see "*Internationalization*" in the *Authentication Node Development Guide*. The following variables can be used in the `VALUE` field: **`{{user}}`** Replaced with the username value of the account registered in the ForgeRock Authenticator app, for example *Demo*. **`{{issuer}}`** Replaced with the issuer value of the account registered in the ForgeRock Authenticator app, for example *ForgeRock*. Example: `Login attempt from {{user}} at {{issuer}}.`
Remove 'skip' option	Enable this option in the node to make the push authentication mandatory. When set to Disabled the user can skip the push authentication requested by the node, and tree evaluation continues along the `Skipped` outcome path. Default: Disabled **Note** Nodes in authentication trees are not affected by the `Two Factor Authentication Mandatory` property, available at Realms > *Realm Name* > Authentication > Settings > General, as it only applies to modules within authentication chains.

Example:

Example Push Tree


Authentication - Trees > Push Example
Save
Start
Username Collector
Polling Wait Node (8 Seconds)
Done
Exited
Push Result Verifier Node
Success
Failure
Expired
Waiting
Push Sender
Sent
Not Registered
Skipped
Set Failure URL (Registration Chain)
Inner Tree Evaluator (Alternative Authentication Tree)
True
False
Success
Failure
Recovery Code Collector Decision (Push Type)
True
False
Retry Limit Decision (3 Attempts)
Retry
Reject

The example tree above shows one possible implementation of multi-factor push authentication.

If the *user has a registered device*:

1. A push notification is sent to their registered device.
2. The Polling Wait Node pauses the authentication tree for 8 seconds, during which time the user can respond to the push notification on their device, for example by using the ForgeRock Authenticator application.
   - If the user responds positively, they are authenticated successfully and logged in.
   - If the user responds negatively, they are not authenticated successfully and do not receive a session.

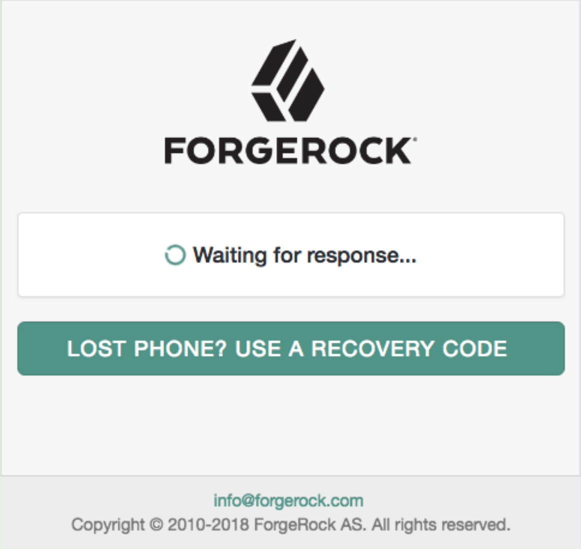
- If the push notification expires, the tree will send a new push notification.

> **Tip**
>
> A Retry Limit Decision node could be used here to constrain the number of times a new code is sent.

- If the user has not yet responded, the tree loops back a step and the Polling Wait Node pauses the authentication tree for another 8 seconds.

If the user exits the Polling Wait Node, they can enter a recovery code in order to authenticate.

> **Tip**
>
> In this situation, configure the Exit Message property in the Polling Wait node with a message such as: `Lost phone? Use a Recovery Code`, which appears as follows:


FORGEROCK
Waiting for response...
LOST PHONE? USE A RECOVERY CODE
info@forgerock.com
Copyright © 2010-2018 ForgeRock AS. All rights reserved.

A Retry Limit Decision node allows three attempts at entering a recovery code before failing the authentication.

If the user *does not have a registered device*:

1. Because trees cannot currently register devices, a Set Failure URL node redirects the user to an authentication chain which can register a device to the user's profile.

2. That registration chain redirects the user back to the push example tree when registration is complete.

If the configuration allows it and the user *chooses to skip multi-factor authentication*:

1. An Inner Tree Evaluator node provides an alternative method of authentication, for example an LDAP Decision node.

2. Depending on the outcome of the inner tree, the push example tree evaluation continues to the `Success` or `Failure` outcome.

Persistent Cookie Decision Node

The Persistent Cookie Decision authentication node checks for the existence of the persistent cookie specified in the Persistent cookie name property, the default being `session-jwt`.

If the cookie is present, the node verifies the signature of the JWT stored in the cookie by using the signing key specified in the HMAC signing key property.

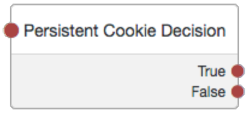
If the signature is valid, the node will decrypt the payload of the JWT by using the key pair specified in the Persistent Cookie Encryption Certificate Alias property. This property can be found at the global level by navigating to Configure > Authentication > Core Attributes > Security, or at the realm level by navigating to Realms > *Realm Name* > Authentication > Settings > Security.

Within the decrypted JSON payload is information such as the UID of the identity, and the client IP address. Enable the Enforce client IP property to verify that the current IP address and the client IP address in the cookie are identical.

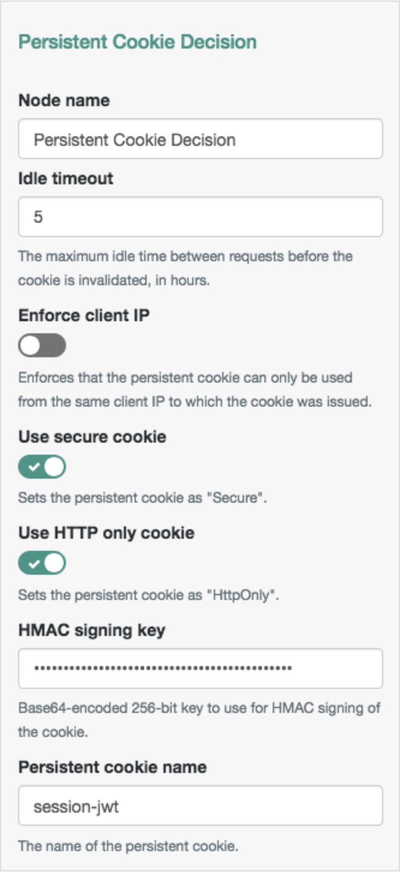
Note

The Persistent Cookie Decision authentication node recreates the received persistent cookie, updating the value for the idle time property. Therefore, cookie creation properties as used by the Set Persistent Cookie Node are also available in the Persistent Cookie Decision authentication node.

The tree evaluation continues along the `True` outcome path if the persistent cookie is present and all the verification checks above are satisfied. Otherwise, tree evaluation continues along the `False` outcome path.



Properties:



Persistent Cookie Decision

Node name: Persistent Cookie Decision

Idle timeout: 5

The maximum idle time between requests before the cookie is invalidated, in hours.

Enforce client IP: ○

Enforces that the persistent cookie can only be used from the same client IP to which the cookie was issued.

Use secure cookie: ✓

Sets the persistent cookie as "Secure".

Use HTTP only cookie: ✓

Sets the persistent cookie as "HttpOnly".

HMAC signing key: ••

Base64-encoded 256-bit key to use for HMAC signing of the cookie.

Persistent cookie name: session-jwt

The name of the persistent cookie.

Property	Usage
Idle Timeout	Specifies the maximum amount of idle time allowed before the persistent cookie is invalidated, in hours. If no requests are received and the time is exceeded, the cookie is no longer valid.
Enforce Client IP	When enabled, ensures that the persistent cookie is only used from the same client IP to which the cookie was issued.
Use secure cookie	When enabled, adds the `Secure` flag to the persistent cookie. If the `Secure` flag is included, the cookie can only be transferred over HTTPS. When a request is made over HTTP, the cookie is not made available to the application.
Use HTTP only cookie	When enabled, adds the `HttpOnly` flag to the persistent cookie. When the `HttpOnly` flag is included, that cookie will not be accessible through JavaScript. According to RFC 6265, the `HttpOnly` flag:

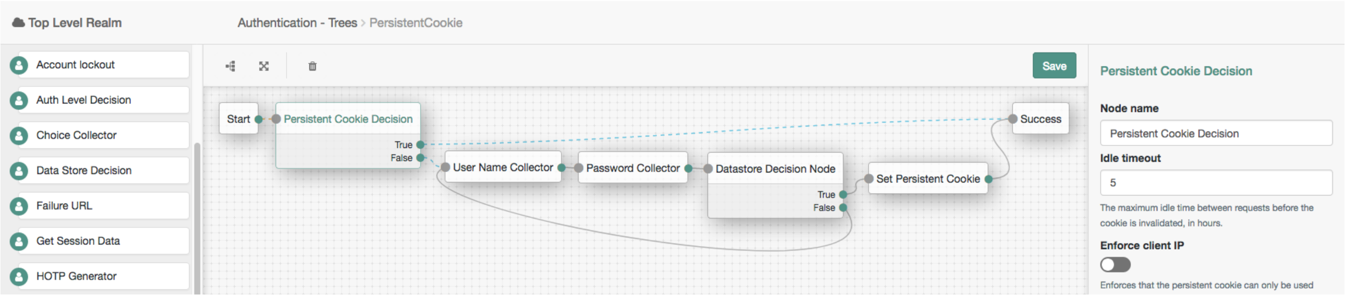
Property	Usage
	instructs the user agent to omit the cookie when providing access to cookies via 'non-HTTP' APIs (for example, a web browser API that exposes cookies to scripts).

HMAC Signing Key	Specifies a key to use for HMAC signing of the persistent cookie. Values must be base64-encoded and at least 256 bits (32 bytes) long.
	Important
	To consume the persistent cookies generated by instances of the Set Persistent Cookie Node in the tree, ensure they are using the same HMAC signing key.
	To generate an HMAC signing key, run one of the following commands:
	`$ openssl rand -base64 32`
	or
	`$ cat /dev/urandom | LC_ALL=C tr -dc 'a-zA-Z0-9' | fold -w 32 | head -n 1| base64`

Persistent cookie name	Specifies the name of the persistent cookie to check.

Example:

PersistentCookie Tree



Persistent Cookie Decision

Node name

Persistent Cookie Decision

Idle timeout

5

The maximum idle time between requests before the cookie is invalidated, in hours.

Enforce client IP

○

Enforces that the persistent cookie can only be used

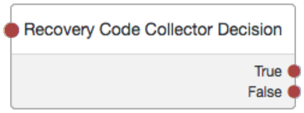
Recovery Code Collector Decision Node

The Recovery Code Collector Decision authentication node allows users to authenticate using a recovery code provided when registering a device for multi-factor authentication.

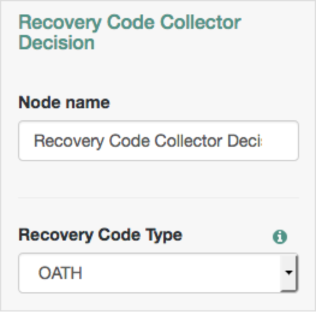
Use this node when a tree is configured to use push notifications or one-time passwords but the user has lost the registered device, and must therefore use an alternative method for authentication. For more information on viewing the recovery codes when registering a device, see "Registering the ForgeRock Authenticator for Multi-Factor Authentication".

Tree evaluation continues along the True outcome path if the provided recovery code matches one belonging to the user. To determine whether the provided code belongs to the user, the tree must have already acquired the username, for example by using a Username Collector Node.

If the recovery code does not match, or a username has not been acquired, tree evaluation continues along the False outcome path.



Properties:



Recovery Code Collector Decision

Node name

Recovery Code Collector Deci

Recovery Code Type

OATH

Property	Usage
Recovery Code Type	Specify the type of recovery code the user will submit for verification. Default: OATH

Recovery Code Display Node

The Recovery Code Display node is used in conjunction with the WebAuthn Registration Node. It retrieves generated recovery codes from the transient state and presents them to the user, for safe-keeping. The codes can be used to authenticate if a registered device is lost or stolen.

Generated recovery codes are inserted into transient state when tree evaluation continues along the Success outcome path of the WebAuthn Registration Node. Connect the Recovery Code Display node to the Success outcome path to display the codes.

If no recovery codes are available in transient state, tree evaluation continues along the only outcome path, and nothing is displayed to the user.

Important

Generated recovery codes cannot be retrieved from the user's profile - they are one-way encrypted. The Recovery Code Display node is the one and only opportunity to view the recovery codes, and keep them safe.

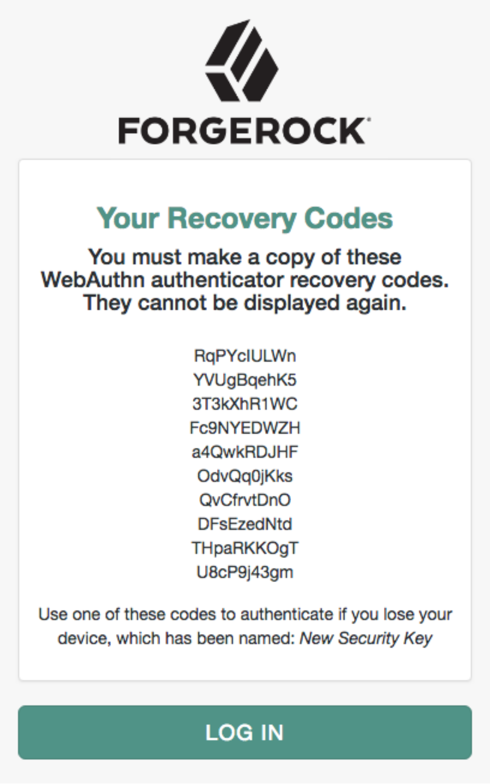
Properties:

This node has no configurable properties.

Example:

The following is an example of the output of the Recovery Code Display node:

Example output of the Recovery Code Display node



FORGEROCK

Your Recovery Codes

You must make a copy of these
WebAuthn authenticator recovery codes.
They cannot be displayed again.

```
RqPYcIULWn
YVUgBqehK5
3T3kXhR1WC
Fc9NYEDWZH
a4QwkRDJHF
OdvQq0jKks
QvCfrvtDnO
DFsEzedNtd
THpaRKKOgT
U8cP9j43gm
```

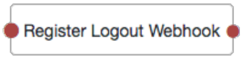
Use one of these codes to authenticate if you lose your device, which has been named: *New Security Key*

Register Logout Webhook Node

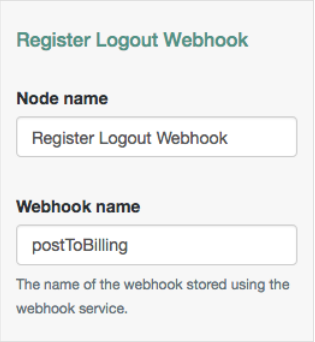
The Register Logout Webhook authentication node registers the specified webhook to trigger when a user's session ends. The webhook triggers when a user explicitly logs out, or the maximum idle time or expiry time of the session is reached.

The webhook is only registered if tree evaluation passes through the Register Logout Webhook node. You can register multiple webhooks during the authentication process, but they must be unique.

For more information on webhooks, see "Configuring Authentication Webhooks".



Properties:



Register Logout Webhook

Node name

Register Logout Webhook

Webhook name

postToBilling

The name of the webhook stored using the webhook service.

Property	Usage
Webhook name	Specify the name of the webhook to register.

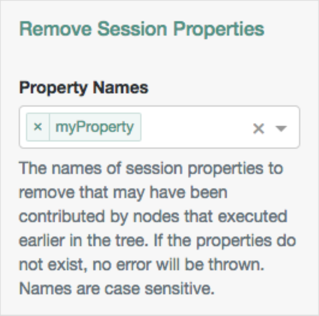
Remove Session Properties Node

The Remove Session Properties authentication node enables the removal of properties from the session. The session properties may have been set by a Set Session Properties node elsewhere in the tree.

If a specified key is not found in the list of session properties that will be added to the session upon successful authentication, no error is thrown and tree evaluation continues along the single outcome path.

If a specified key is found, the tree evaluation continues along the single outcome path after setting the value of the property to `null`.

Properties:



Remove Session Properties

Property Names

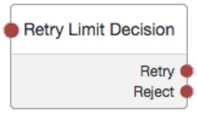
× myProperty

The names of session properties to remove that may have been contributed by nodes that executed earlier in the tree. If the properties do not exist, no error will be thrown. Names are case sensitive.

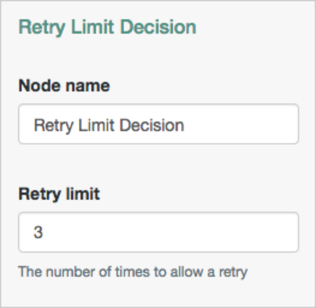
Property	Usage
Property Names	Enter one or more key names of properties to remove from the session.

Retry Limit Decision Node

The Retry Limit Decision authentication node allows the specified number of passes through to the `Retry` outcome path, before continuing tree evaluation along the `Reject` outcome path.



Properties:



Retry Limit Decision

Node name

Retry Limit Decision

Retry limit

3

The number of times to allow a retry

Property	Usage
Retry limit	Specify the number of times to allow a retry. Default: `3`
Save Retry Limit to User	Specify whether the number of failed login attempts persists between successful authentications. Possible values are: • Disabled

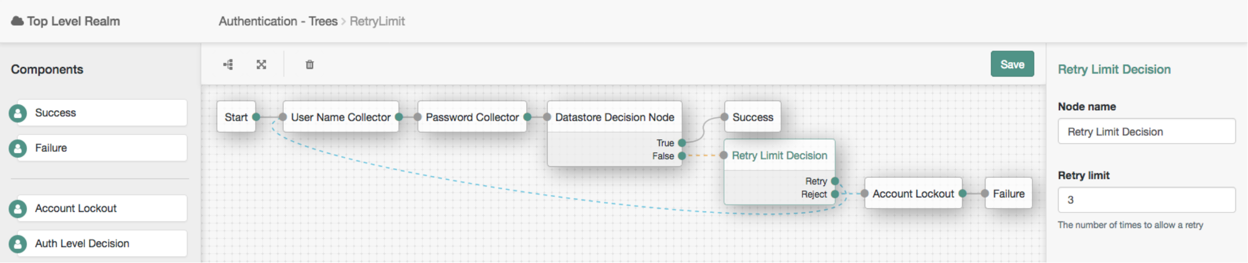
Property	Usage

	This is the default value. The node saves the number of failed login attempt in the tree's `nodeRetryLimitKey` shared state property, which is discarded when the authentication session ends.
	For security reasons, ForgeRock recommends that you enable this setting.
	• Enabled
	The node saves the number of failed login attempts to the user's profile. New authentication journeys using the Retry Limit Decision node will use the stored value as the starting point for the retry limit.
	AM resets the count after the user authenticates successfully with a tree that contains this node.
	If AM cannot find the user's profile, the authentication journey will end with an error.
	Important
	After upgrading to AM 6.5.4 or later, you must update the identity store's schema manually before enabling this feature.
	+ *How Do I Apply the New Schema?*
	To update the identity store schema for the Retry Limit Decision Node, perform the following steps:
	1. Change directories to the path where you deployed the `openam.war` file. For example, `/path/to/tomcat/webapps/openam`.
	2. Locate the `opendj_retry_limit_node_count.ldif` file in the `WEB-INF/template/ldif/opendj` path.
	3. Update the identity store schema using the LDIF file. For example:
	`$`
	`/path/to/opendj/bin/ldapmodify \`
	`--hostname 'id.example.com' \`
	`--port 1636 \`
	`--useSsl \`
	`--usePkcs12TrustStore /path/to/opendj/config/keystore \`
	`--trustStorePasswordFile /path/to/opendj/config/keystore.pin \`
	`--continueOnError \`
	`--bindDN uid=admin \`
	`--bindPassword ${ds.admin.password} \`
	`/path/to/tomcat/webapps/openam/WEB-INF/template/ldif/opendj/`
	`opendj_retry_limit_node_count.ldif`

Property	Usage
	Now you are ready to enable the Save Retry Limit to User switch in the "Retry Limit Decision Node".

Example:

RetryLimit Tree



Top Level Realm

Authentication - Trees > RetryLimit

Retry Limit Decision

Node name: Retry Limit Decision

Retry limit: 3

The number of times to allow a retry

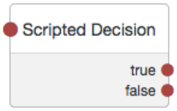
Scripted Decision Node

The Scripted Decision authentication node allows execution of scripts during authentication. Tree evaluation continues along the path matching the result.

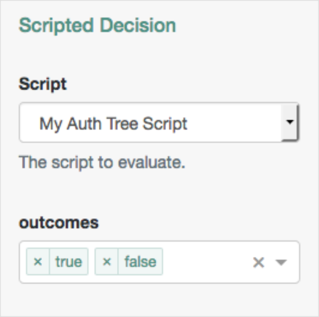
The script defines the possible outcome paths by setting one or more values of a string variable named `outcome`. For more information on creating scripts, see "Managing Scripts With the AM Console".

The tree evaluation continues along the outcome path that matches the value of the `outcome` variable when script execution completes.

For information about the API available for use in the Scripted Decision Node, see "Scripted Decision Node API Functionality".



Properties:



Scripted Decision

Script: My Auth Tree Script

The script to evaluate.

outcomes: true, false

Property	Usage
Script	Select the script to execute from the drop-down field.
outcomes	Enter the possible strings that can be assigned to the `outcome` variable by the script. These strings provide the possible outcome paths the tree can continue along.

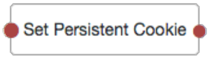
Set Persistent Cookie Node

The Set Persistent Cookie authentication node creates a persistent cookie named after the value specified in the Persistent cookie name property, the default being `session-jwt`.

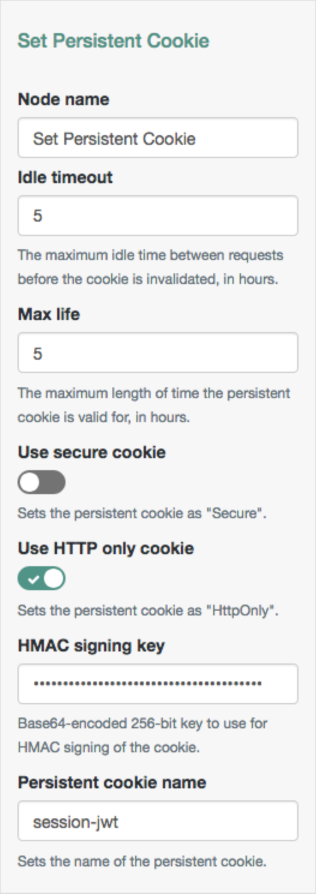
The cookie contains a JWT, inside which there is a JSON payload with information such as the UID of the identity, and the client IP address.

The node encrypts the JWT using the key pair specified in the Persistent Cookie Encryption Certificate Alias property. This property can be found by navigating to Configure > Authentication > Core Attributes > Security.

The node signs the cookie with the signing key specified in the HMAC signing key property. Any node that will read the persistent cookie must be configured with the same HMAC signing key.



Properties:



Set Persistent Cookie

Node name: Set Persistent Cookie

Idle timeout: 5

The maximum idle time between requests before the cookie is invalidated, in hours.

Max life: 5

The maximum length of time the persistent cookie is valid for, in hours.

Use secure cookie: ○ (off)

Sets the persistent cookie as "Secure".

Use HTTP only cookie: ✓ (on)

Sets the persistent cookie as "HttpOnly".

HMAC signing key: ••

Base64-encoded 256-bit key to use for HMAC signing of the cookie.

Persistent cookie name: session-jwt

Sets the name of the persistent cookie.

Property	Usage
Idle Timeout	Specifies the maximum amount of idle time allowed before the persistent cookie is invalidated, in hours. If no requests are received and the time is exceeded, the cookie is no longer valid.
Max life	Specifies the length of time the persistent cookie remains valid, in hours. If that time is exceeded, the cookie is no longer valid.
Use secure cookie	When enabled, adds the `Secure` flag to the persistent cookie. If the `Secure` flag is included, the cookie can only be transferred over HTTPS. When a request is made over HTTP, the cookie is not made available to the application.
Use HTTP only cookie	When enabled, adds the `HttpOnly` flag to the persistent cookie. When the `HttpOnly` flag is included, that cookie will not be accessible through JavaScript. According to RFC 6265, the `HttpOnly` flag:

Property	Usage
	instructs the user agent to omit the cookie when providing access to cookies via 'non-HTTP' APIs (for example, a web browser API that exposes cookies to scripts).

HMAC Signing Key	Specifies a key to use for HMAC signing of the persistent cookie. Values must be base64-encoded and at least 256 bits (32 bytes) long.
	Important
	To consume the persistent cookies generated by instances of the Set Persistent Cookie Node in the tree, ensure they are using the same HMAC signing key.
	To generate an HMAC signing key, run one of the following commands:
	`$ openssl rand -base64 32`
	or
	`$ cat /dev/urandom | LC_ALL=C tr -dc 'a-zA-Z0-9' | fold -w 32 | head -n 1| base64`

Persistent cookie name	Specifies the name used for the persistent cookie.

Set Session Properties Node

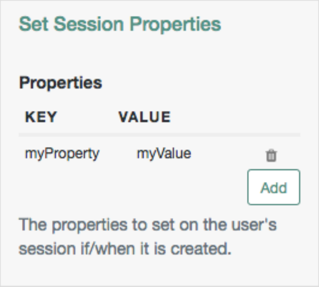
The Set Session Properties authentication node allows the addition of key:value properties to the user's session if authentication is successful.

Tip

You can access session properties using a variable in a webhook. For more information, see "Configuring Authentication Webhooks".

The tree evaluation continues along the single outcome path after setting the specified properties in the session.

Properties:



Set Session Properties

Properties

KEY	VALUE
myProperty	myValue

The properties to set on the user's session if/when it is created.

Property	Usage
Properties	To add a session property, select the Add button, enter a key name and a value, and then select the plus icon. Repeat the steps to add multiple properties.

State Metadata Node

The State Metadata authentication node returns selected attributes from the shared state as metadata.

The node sends a `MetaDataCallback` to retrieve shared state values which are added to the JSON response from the `/authenticate` endpoint. This example shows how a shared state attribute, `mail`, is returned in the JSON output:

```
"callbacks": [
    {
        "type": "MetadataCallback",
        "output": [
            {
                 "name": "data",
                 "value":
                     {
                         "mail": "bjensen@example.com"
                     }
            }
        ]
    }
]
```

You can use the State Metadata node when you want to display custom information that includes user attributes, without having to alter the existing authentication journey.

For example, for OTP authentication with a choice of email or SMS, create a State Metadata node to return the user's email address or phone number. These attributes can be used in conjunction with an OTP Collector Decision Node and, optionally, a Scripted Decision Node, to customize the data for display later in the journey.

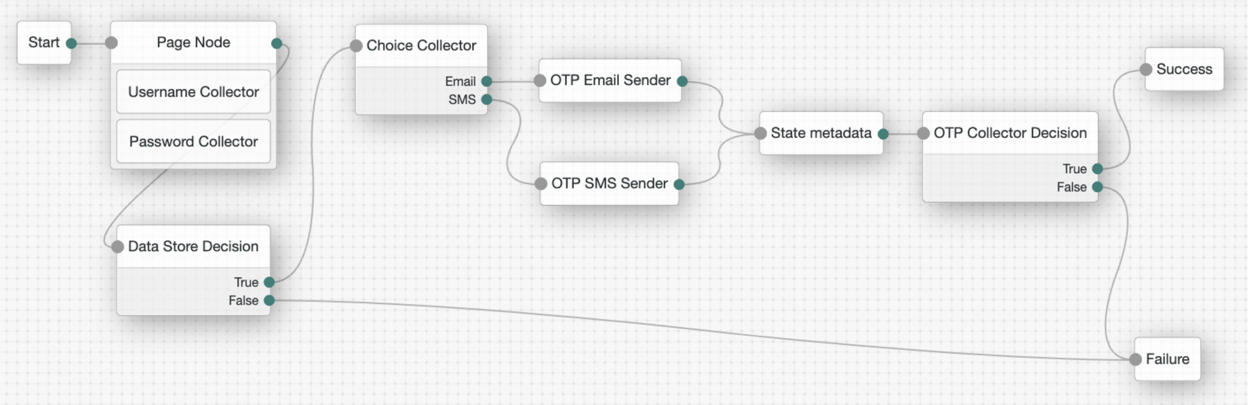
Tree evaluation continues along the single outcome path after the callback.

Properties:

Property	Usage
Attributes	Specify one or more shared state attribute names for return.

Example:

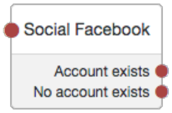
State Metadata Tree



Social Facebook Node

The Social Facebook authentication node is a duplicate of the OAuth 2.0 Node node, preconfigured to work with Facebook. Only the `Client ID` and `Client Secret` are required to be populated.

The tree evaluation continues along the `Account Exists` path if an account matching the attributes retrieved from Facebook are found in the user data store. Otherwise, the tree evaluation continues along the `No account exists` path.



Properties:

Property	Usage
Client ID	Specifies the `client_id` parameter as provided by Facebook.
Client Secret	Specifies the `client_secret` parameter as provided by Facebook.
Authentication Endpoint URL	Specifies the URL to the social provider's endpoint handling authentication as described in section 3.1 of The OAuth 2.0 Authorization Framework (RFC 6749).

Property	Usage
	Default: `https://www.facebook.com/dialog/oauth`
Access Token Endpoint URL	Specifies the URL to the endpoint handling access tokens as described in section 3.2 of The OAuth 2.0 Authorization Framework (RFC 6749). Default: `https://graph.facebook.com/v2.12/oauth/access_token`
User Profile Service URL	Specifies the user profile URL that returns profile information. Default: `https://graph.facebook.com/v2.6/me?fields=name%2Cemail%2Cfirst_name%2Clast_name`
OAuth Scope	Specifies a comma-separated list of user profile attributes that the client application requires, according to *The OAuth 2.0 Authorization Framework (RFC 6749)* . The list depends on the permissions that the resource owner, such as the end user, grants to the client application.
Redirect URL	Specifies the URL the user is redirected to by Facebook after authenticating, to continue the authentication tree flow. Set this property to the URL of the AM XUI, for example `https://openam.example.com:8443/openam/XUI/`. **Tip** If the tree is not in the top-level realm, you can specify the realm in the redirect URL. Use a DNS alias for the realm, or add the realm as a query parameter, for example `https://openam.example.com:8443/openam/XUI/?realm=/mySubRealm`. For more information, see "To Configure DNS Aliases for Accessing a Realm" in the *Setup and Maintenance Guide*.
Social Provider	Specifies the name of the social provider for which this node is being set up. Default: `facebook`
Auth ID Key	Specifies the attribute the social identity provider uses to identify an authenticated individual. Default: `id`
Use Basic Auth	Specifies that the client uses HTTP Basic authentication when authenticating to the social provider. Default: `true`
Account Provider	Specifies the name of the class that implements the account provider. Default: `org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider`
Account Mapper	Specifies the name of the class that implements the method of locating local accounts based on the attributes returned from Facebook. Default:

Property	Usage
	`org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper`
Attribute Mapper	Specifies the list of fully qualified class names for implementations that map attributes from Facebook to AM profile attributes. Default: `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper|uid|facebook-`

Account Mapper Configuration	Specifies the attribute configuration used to map the account of the user authenticated in the Social Facebook provider to the local data store in AM. Valid values are in the form *`provider-attr=local-attr`*.
	Default: `id=uid`.
	Tip
	When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings, by using dot notation.
	For example, given a JSON payload of:
	`{`
	`  "sub" : "12345",`
	`  "name" : {`
	`    "first_name" : "Demo",`
	`    "last_name" : "User"`
	`  }`
	`}`
	You can create a mapper such as:
	`name.first_name=cn`

Attribute Mapper Configuration	Map of Facebook user account attributes to local user profile attributes, with values in the form *`provider-attr=local-attr`*. Default: `name=cn`, `last_name=sn`, `id=uid`, `first_name=givenname`, `email=mail`. **Tip** When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings, by using dot notation. For example, given a JSON payload of:

Property	Usage

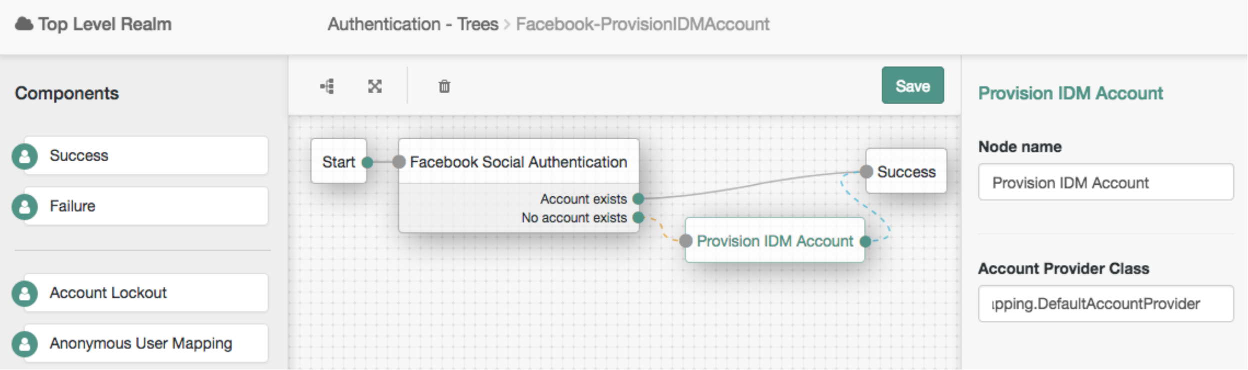
	`{`
	`  "sub" : "12345",`
	`  "name" : {`
	`    "first_name" : "Demo",`
	`    "last_name" : "User"`
	`  }`
	`}`
	You can create a mapper such as:
	`name.first_name=cn`

Save attributes in the session	When enabled, saves the attributes in the Attribute Mapper Configuration field to the AM session. Default: `true`.
OAuth 2.0 Mix-Up Mitigation Enabled	Controls whether the authentication node carries out additional verification steps when it receives the authorization code from the authorization server. Specifies that the client must compare the issuer identifier of the authorization server upon registration with the issuer value returned in the `iss` response parameter. If they do not match, the client must abort the authorization process. The client must also confirm that the authorization server's response is intended for the client by comparing the client's client identifier to the value of the `client_id` response parameter. The Token Issuer property must be entered when the OAuth 2.0 Mix-Up Mitigation feature is enabled, so that the validation can succeed. The authorization code response will contain an issuer value (`iss`) that will be validated by the client. **Note** Consult with the authorization server's documentation on what value it uses for the issuer field. For more information, see section 4 of OAuth 2.0 Mix-Up Mitigation Draft.
Token Issuer	Corresponds to the expected issuer identifier value in the `iss` field of the ID token. Example: `https://graph.facebook.com`

Example:

The following example uses the Provision IDM Account authentication node to allow users who have performed social authentication using Facebook to provision an account using IDM, if they do not have a matching existing profile.

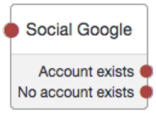
Facebook-ProvisionIDMAccount Tree With Provision IDM Account Node



Social Google Node

The Social Google authentication node is a duplicate of the OAuth 2.0 Node node, preconfigured to work with Google. Only the `Client ID` and `Client Secret` are required to be populated.

The tree evaluation continues along the `Account Exists` path if an account matching the attributes retrieved from Google are found in the user data store. Otherwise, the tree evaluation continues along the `No account exists` path.



Properties:

Property	Usage
Client ID	Specifies the `client_id` parameter as provided by Google.
Client Secret	Specifies the `client_secret` parameter as provided by Google.
Authentication Endpoint URL	Specifies the URL to the social provider's endpoint handling authentication as described in section 3.1 of The OAuth 2.0 Authorization Framework (RFC 6749). Default: `https://accounts.google.com/o/oauth2/v2/auth`
Access Token Endpoint URL	Specifies the URL to the endpoint handling access tokens as described in section 3.2 of The OAuth 2.0 Authorization Framework (RFC 6749). Default: `https://www.googleapis.com/oauth2/v4/token`
User Profile Service URL	Specifies the user profile URL that returns profile information. Default: `https://www.googleapis.com/oauth2/v3/userinfo`

Property	Usage
OAuth Scope	Specifies a space-separated list of user profile attributes that the client application requires, according to *The OAuth 2.0 Authorization Framework (RFC 6749)* . The list depends on the permissions that the resource owner, such as the end user, grants to the client application. Default: `profile email`.
Redirect URL	Specifies the URL the user is redirected to by Google after authenticating, to continue the authentication tree flow. Set this property to the URL of the AM XUI, for example `https://openam.example.com:8443/openam/XUI/`. **Tip** If the tree is not in the top-level realm, you can specify the realm in the redirect URL. Use a DNS alias for the realm, or add the realm as a query parameter, for example `https://openam.example.com:8443/openam/XUI/?realm=/mySubRealm`. For more information, see "To Configure DNS Aliases for Accessing a Realm" in the *Setup and Maintenance Guide*.
Social Provider	Specifies the name of the social provider for which this node is being set up. Default: `google`
Auth ID Key	Specifies the attribute the social identity provider uses to identify an authenticated individual. Default: `sub`
Use Basic Auth	Specifies that the client uses HTTP Basic authentication when authenticating to Google. Default: `true`
Account Provider	Specifies the name of the class that implements the account provider. Default: `org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider`
Account Mapper	Specifies the name of the class that implements the method of locating local accounts based on the attributes returned from Google. Default: `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper`
Attribute Mapper	Specifies the list of fully qualified class names for implementations that map attributes from Google to AM profile attributes. Default:

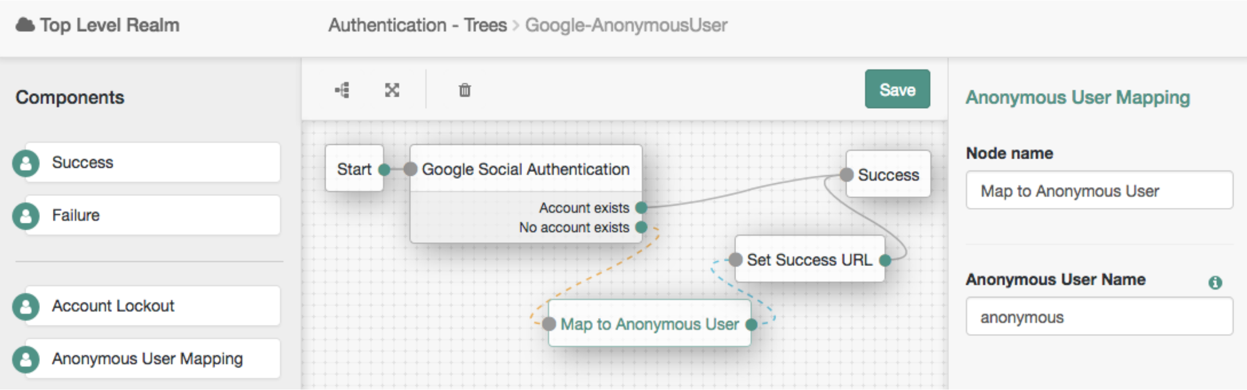
<table>
<tr><th>Property</th><th>Usage</th></tr>
<tr><td></td><td><code>org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper|iplanet-am-user-alias-list|google-</code></td></tr>
<tr><td>Account Mapper Configuration</td><td>
<p>Specifies the attribute configuration used to map the account of the user authenticated in the Social Google provider to the local data store in AM. Valid values are in the form <em><code>provider-attr=local-attr</code></em>.</p>
<p>Default: <code>sub=uid</code>.</p>
<p><strong>Tip</strong></p>
<p>When using the <code>org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper</code> class, you can parse JSON objects in mappings, by using dot notation.</p>
<p>For example, given a JSON payload of:</p>
<pre>
{
  "sub" : "12345",
  "name" : {
    "first_name" : "Demo",
    "last_name" : "User"
  }
}
</pre>
<p>You can create a mapper such as:</p>
<p><code>name.first_name=cn</code></p>
</td></tr>
<tr><td>Attribute Mapper Configuration</td><td>
<p>Map of Google user account attributes to local user profile attributes, with values in the form <em><code>provider-attr=local-attr</code></em>.</p>
<p>Default: <code>sub=uid</code>, <code>name=cn</code>, <code>given_name=givenName</code>, <code>family_name=sn</code>, <code>email=mail</code>.</p>
<p><strong>Tip</strong></p>
<p>When using the <code>org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper</code> class, you can parse JSON objects in mappings, by using dot notation.</p>
<p>For example, given a JSON payload of:</p>
<pre>
{
  "sub" : "12345",
  "name" : {
    "first_name" : "Demo",
    "last_name" : "User"
  }
}
</pre>
<p>You can create a mapper such as:</p>
</td></tr>
</table>

Property	Usage
	`name.first_name=cn`
Save attributes in the session	When enabled, saves the attributes in the Attribute Mapper Configuration field to the AM session. Default: `true`.
OAuth 2.0 Mix-Up Mitigation Enabled	Controls whether the authentication node carries out additional verification steps when it receives the authorization code from the authorization server. Specifies that the client must compare the issuer identifier of the authorization server upon registration with the issuer value returned in the `iss` response parameter. If they do not match, the client must abort the authorization process. The client must also confirm that the authorization server's response is intended for the client by comparing the client's client identifier to the value of the `client_id` response parameter. The Token Issuer property must be entered when the OAuth 2.0 Mix-Up Mitigation feature is enabled, so that the validation can succeed. The authorization code response will contain an issuer value (`iss`) that will be validated by the client. **Note** Consult with the authorization server's documentation on what value it uses for the issuer field. For more information, see section 4 of OAuth 2.0 Mix-Up Mitigation Draft.
Token Issuer	Corresponds to the expected issuer identifier value in the `iss` field of the ID token. Example: `https://accounts.google.com`

Example:

The following example uses the Anonymous User Mapping authentication node to allow users who have performed social authentication using Google to access AM as an anonymous user if they do not have a matching existing profile.

Google-AnonymousUser Tree With Anonymous User Mapping Node



Node name: Map to Anonymous User

Anonymous User Name: anonymous

Social Ignore Profile Node

The Social Ignore Profile authentication node specifies if a local user profile should be ignored. If tree evaluation passes through this node, after successful social authentication, AM issues an SSO token regardless of whether a user profile exists in the data store. The presence of a user profile is not checked.

Properties:

This node has no configurable properties.

Success URL Node

The Success URL authentication node sets the URL to be redirected to when authentication succeeds.

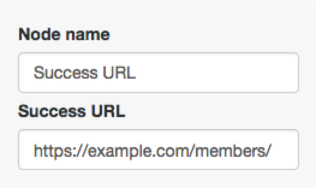
> **Note**
>
> Specifying a success URL in a tree overrides any `goto` query string parameters.

For more information on how AM determines the redirection URL, and to configure the Validation Service to trust redirection URLs, see "Configuring Success and Failure Redirection URLs".

Tip

The URL is also saved into the `sharedState` object, under a property named `successUrl`, which can be useful for custom node developers. For more information, see "Customizing Authentication Trees".

Properties:



Node name

Success URL

Success URL

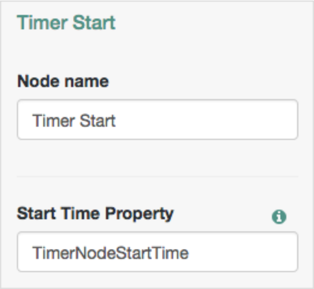
https://example.com/members/

Property	Usage
Success URL	Specify the full URL to be redirected to when the authentication succeeds.

Timer Start Node

The Timer Start authentication node starts a named timer metric, which can be stopped elsewhere in the tree by using the Timer Stop Node.

Properties:



Timer Start

Node name

Timer Start

Start Time Property

TimerNodeStartTime

Property	Usage
Start Time Property	Specify a property name into which to store the current time. Specify the same value in any instances of the Timer Stop Node that measure the time elapsed since tree evaluation passed through this node.

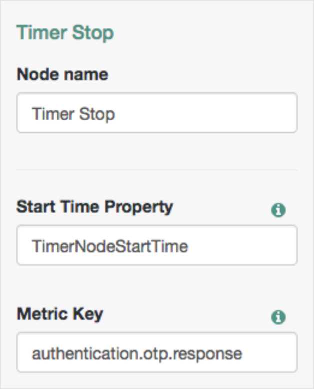
Timer Stop Node

The Timer Stop authentication node records the time elapsed since tree evaluation passed through the specified Timer Start Node in the specified metric name. For information on the `Timer` metric type, see "Monitoring Metric Types" in the *Setup and Maintenance Guide*.

Note that the time stored in the specified `Start Time Property` property is not reset by the Timer Stop Node, so other Timer Stop Nodes in the tree can also calculate the time elapsed since tree evaluation passed through the same Timer Start Node.

The metric is exposed in all available interfaces, as described in "Monitoring Interfaces" in the *Setup and Maintenance Guide*.

Properties:



Timer Stop

Node name: Timer Stop

Start Time Property: TimerNodeStartTime

Metric Key: authentication.otp.response

Property	Usage
Start Time Property	Specify the property name containing the time from which to calculate the elapsed time.
Metric Key	Specify the name of a metric in which to store the calculated elapsed time.

Username Collector Node

The Username Collector authentication node prompts the user to enter their username.

The tree evaluation continues along the single outcome path after capturing the username.

Properties:

This node has no configurable properties.

WebAuthn Authentication Node

The WebAuthn Authentication node allows users of supported clients to use a registered FIDO device during authentication.

To determine whether the user has a registered device, the tree must have already acquired a username, for example by using a Username Collector Node.

If the user's client does not support web authentication, tree evaluation will continue along the `Unsupported` outcome path. For example, clients connected over the HTTP protocol rather than HTTPS do not support WebAuthn. [1]

If the user does not have a registered device, tree evaluation continues along the `No Device Registered` outcome path.

If AM encounters an issue when attempting to authenticate using the device, tree evaluation continues along the `Failure` outcome path. For example, AM could not verify that the response from the authenticator was appropriate for the specific instance of the authentication ceremony.

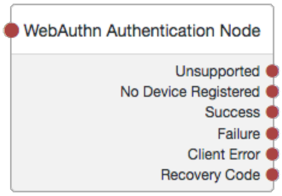
If the user's client encounters an issue when attempting to authenticate using the device, for example, if the timeout was reached, then tree evaluation continues along the `Client Error` outcome path. This outcome is used whenever the client throws a `DOMException`, as required by the Web Authentication: An API for accessing Public Key Credentials Level 1 specification.

Tip

If a client error occurs, the error type and description are added to a property named `WebAuthenticationDOMException` in the shared state. This property can be read by other nodes later in the tree, if required.

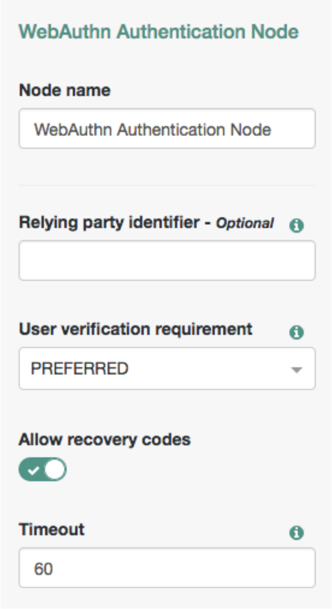
If the Allow recovery code property is enabled, AM provides the user the option to enter a recovery code rather than authenticate using a device. Tree evaluation continues along the `Recovery Code` outcome path if the users chooses to enter a recovery code. To accept and verify the recovery code, ensure the outcome path leads to a Recovery Code Collector Decision Node.

If the user successfully authenticates with a device of the type determined by the User verification requirement property, tree evaluation continues along the `Success` outcome path.



[1] HTTPS may not be required when testing locally, on `http://localhost`, for example. For more information, see Is origin potentially trustworthy?.

Properties:



WebAuthn Authentication Node

Node name: WebAuthn Authentication Node

Relying party identifier - *Optional*:

User verification requirement: PREFERRED

Allow recovery codes: ✓

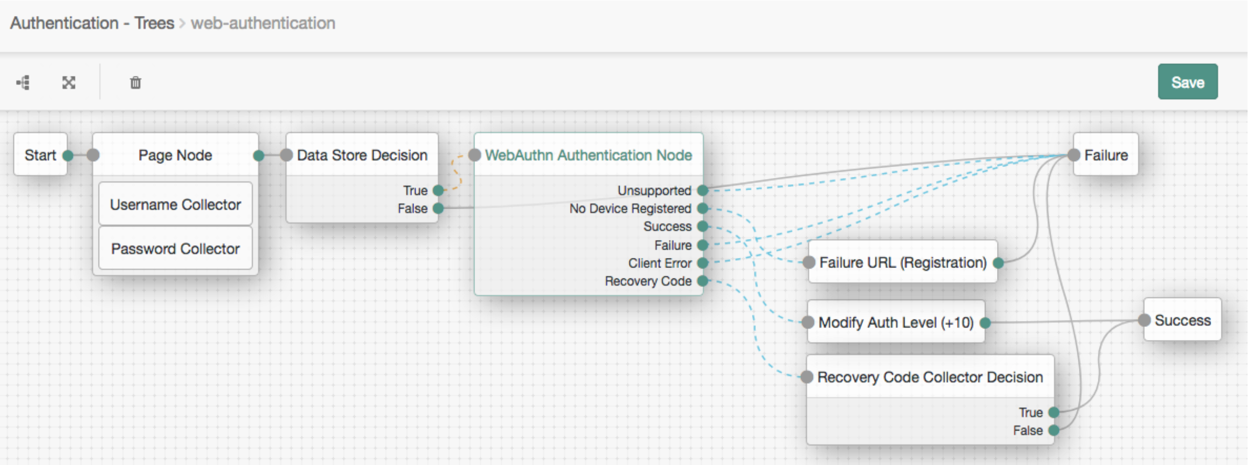
Timeout: 60

Property	Usage
Relying party identifier	Specifies the domain used as the relying party identifier during web authentication. If not specified, AM uses the domain name of the instance, for example `openam.example.com`. Specify an alternative domain if your AM instances are behind a load balancer, for example.
User verification requirement	Specifies the required level of user verification. The available options are: **`REQUIRED`** The authenticator used must verify the identity of the user, for example, by using biometrics. Authenticators that do not verify the identity of the user should not be activated for authentication. **`PREFERRED`** Use of an authenticator that verifies the identity of the user is preferred, but if none are available any authenticator is accepted. **`DISCOURAGED`** Use of an authenticator that verifies the identity of the user is not required. Authenticators that do not verify the identity of the user should be preferred.

Property	Usage
Allow recovery codes	Specify whether to allow the user to enter one of their recovery codes instead of performing an authentication gesture. Enabling this options adds a `Recovery Code` outcome path to the node. This outcome path should lead to a Recovery Code Collector Decision Node in order to collect and verify the recovery code.
Timeout	Specify the number of seconds to wait for a response from an authenticator. If the specified time is reached, tree evaluation continues along the `Client error` outcome path, and a relevant message is stored in the `WebAuthenticationDOMException` property of the shared state.

Example:

Example WebAuthn Authentication Tree



The example tree above shows one possible implementation of a tree for authenticating with WebAuthn devices.

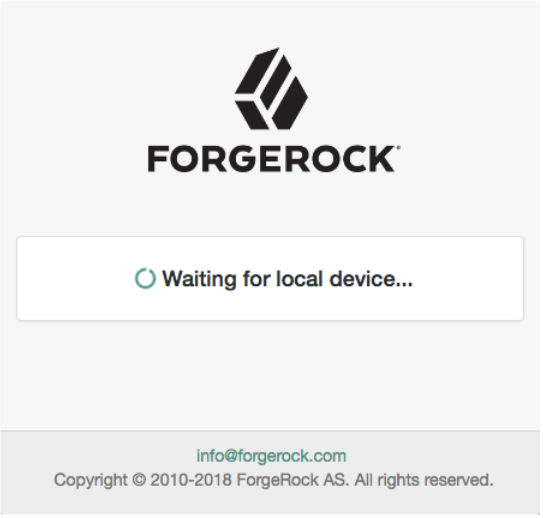
After verifying the users credentials against the configured data store, tree evaluation continues to the WebAuthn Authentication Node.

If the user's client does not support web authentication, the tree fails and the user does not get a session. A more user-friendly approach would be to set a success URL to redirect the user to a page explaining the benefits of multi-factor authentication, and then proceeding to the `Success` node.

If there are no registered WebAuthn devices present in the user's profile, the failure URL is set, pointing to a tree that allows the user to register a device. This stage could also be an Inner Tree Evaluator, with a registration tree inside.

If the user's client does support web authentication, and the connection is secured with TLS, the user will be asked to complete an authorization gesture, for example scanning a fingerprint, or entering a PIN number:

The WebAuthn Authentication node waiting for an authenticator.



Waiting for local device...

info@forgerock.com
Copyright © 2010-2018 ForgeRock AS. All rights reserved.

The user's browser may present a consent pop-up to allow access to the authenticators available on the client. When consent has been granted, the browser activates the relevant authenticators, ready for authentication.

Tip

The relying party details configured in the node are often included in the consent message to help the user verify the entity that is requesting access.

The authenticators the client activates for authentication depends in the value of the properties in the node. For example, if the `User verification requirement` property is set to `REQUIRED`, the client SHOULD only activate authenticators which verify the identity of the user. For extra protection, AM WILL verify that the response from an authenticator matches the criteria configured for the node, and will reject - by using the `Failure` outcome - an authentication attempt by an inappropriate authenticator type.

When the user completes an authorization gesture, for example scanning a fingerprint, or entering a PIN number, tree evaluation continues along the `Success` outcome path. In this example, their authentication level is increased by ten to signify the stronger authentication that has occurred, and the user is taken to their profile page.

If the user clicks the `Use Recovery Code` button, tree evaluation continues to the Recovery Code Collector Decision Node, ready to accept the recovery code. If verified, the user is taken to their profile page.

Any problems encountered during the authentication (thorugh the `Failure` outcome), including a timeout (through the `Client Error` outcome), results in the overall failure of the authentication tree.

WebAuthn Registration Node

The WebAuthn Registration authentication node allows users of supported clients to register FIDO devices for use during authentication.

If the user's client does not support web authentication, tree evaluation will continue along the `Unsupported` outcome path. For example, clients connected over the HTTP protocol rather than HTTPS do not support WebAuthn.[1]

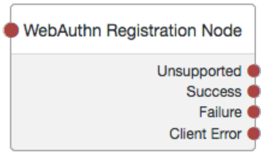
If AM encounters an issue when attempting to register using a device, tree evaluation continues along the `Failure` outcome path. For example, AM could not verify that the response from the authenticator was appropriate for the specific instance of the authentication ceremony.

If the user's client encounters an issue when attempting to register using a device, for example, if the timeout was reached, then tree evaluation continues along the `Client Error` outcome path. This outcome is used whenever the client throws a `DOMException`, as required by the Web Authentication: An API for accessing Public Key Credentials Level 1 specification.

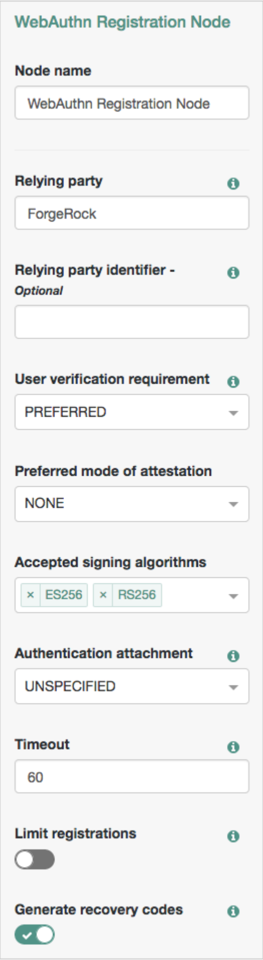
Tip

If a client error occurs, the error type and description are added to a property named `WebAuthenticationDOMException` in the shared state. This property can be read by other nodes later in the tree, if required.

If the user successfully registers an authenticator of the correct determined by the node's properties, tree evaluation continues along the `Success` outcome path.



Properties:



WebAuthn Registration Node

Node name: WebAuthn Registration Node

Relying party: ForgeRock

Relying party identifier - *Optional*:

User verification requirement: PREFERRED

Preferred mode of attestation: NONE

Accepted signing algorithms: ES256, RS256

Authentication attachment: UNSPECIFIED

Timeout: 60

Limit registrations: off

Generate recovery codes: on

Property	Usage
Relying party	Specify the name of the relying party entity that is registering and authenticating users by using web authentication. For example, `Example Inc.`.
Relying party identifier	Specifies the domain used as the relying party identifier during web authentication. If not specified, AM uses the the domain name of the instance, for example `openam.example.com`. Specify an alternative domain if your AM instances are behind a load balancer, for example.
User verification requirement	Specifies the required level of user verification.

Property	Usage

	The available options are:
	`REQUIRED`
	The authenticator used must verify the identity of the user, for example by using biometrics. Authenticators that do not verify the identity of the user should not be activated for registration.
	`PREFERRED`
	Use of an authenticator that verifies the identity of the user is preferred, but if none are available any authenticator is accepted.
	`DISCOURAGED`
	Use of an authenticator that verifies the identity of the user is not required. Authenticators that do not verify the identity of the user should be preferred.

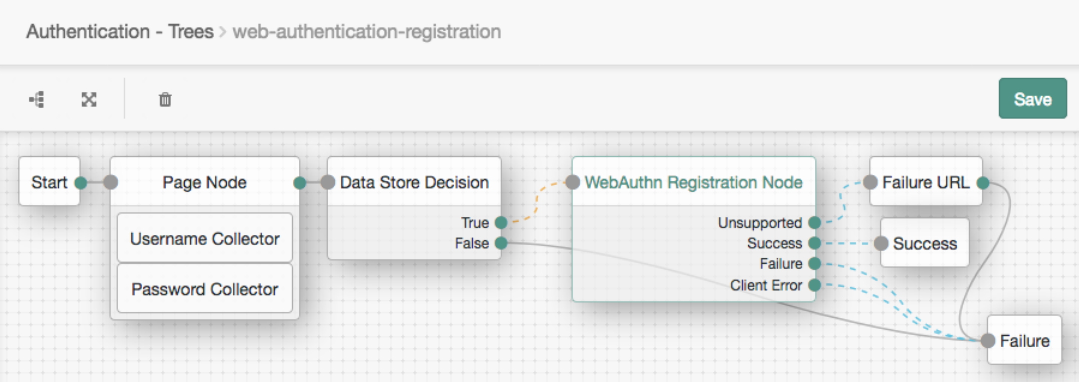
Preferred mode of attestation	Specifies whether AM requires that the authenticator provides attestation statements.
	The available options are:
	`NONE`
	AM does not require the authenticator to provide attestation statements. If the authenticator does send attestation statements, AM *will not* verify them, and will not fail the process.
	`INDIRECT`
	AM does not require the authenticator to provide attestation statements. If the authenticator does send attestation statements, AM *will* verify them, and will fail the process if they fail verification.
	`DIRECT`
	AM requires that the authenticator provides attestation statements, and *will* verify them. The process will fail if the attestation statements cannot be verified.
	AM supports the following attestation formats:
	• Packed
	• FIDO U2F

Property	Usage
	Important You must set the Preferred mode of attestation property to `NONE` to use an authenticator that provides attestation statements in a format other than those above.
Accepted signing algorithms	Specify the algorithms that authenticators can use to sign their assertions.
Authentication attachment	Specifies whether AM requires that the authenticator is a particular attachment type. There are two types of authenticator attachment: • An authenticator that is built-in to the client device is labelled a *platform attachment*. A fingerprint scanner built-in to a phone or laptop is an example of a platform attachment authenticator. • An authenticator that can roam, or move, between different client devices is labelled a *cross-platform attachment*. A USB hardware security key is an example of a cross-platform attachment authenticator. The available options are: **`UNSPECIFIED`** AM accepts any attachment type. **`PLATFORM`** The authenticator must be a *platform* attachment type. The client should not activate other authenticator types for registration. **`CROSS_PLATFORM`** The authenticator must be a *cross-platform* attachment type. The client should not activate other authenticator types for registration.
Timeout	Specify the number of seconds to wait for a response from an authenticator. If the specified time is reached, tree evaluation continues along the `Client error` outcome path, and a relevant message is stored in the `WebAuthenticationDOMException` property of the shared state.
Limit registrations	Specify whether the same authenticator can be registered multiple times. If enabled, the client should not activate an authenticator that is already registered for registration.

Property	Usage
Generate recovery codes	Specify whether web authentication-specific recovery codes should be generated. If enabled, recovery codes are generated and stored in transient state if registration was successful. Use the Recovery Code Display Node to display the codes to the user for safe-keeping. **Important** Generating recovery codes will overwrite all existing web authentication-specific recovery codes. Only the most recent set of recovery codes can be used for authentication if a device has been lost or stolen.

Example:

Example Web Authentication Registration Tree


Authentication - Trees > web-authentication-registration

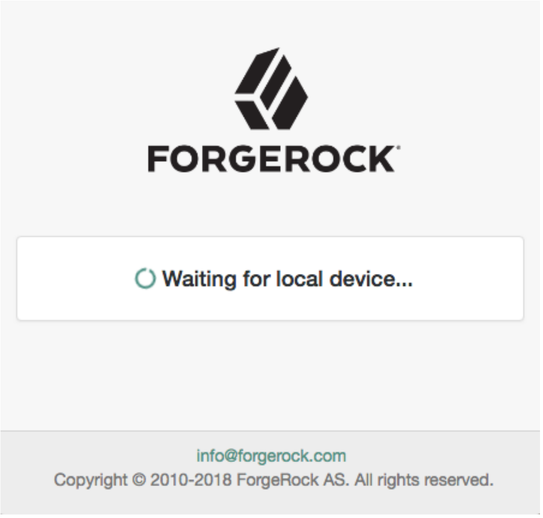
The example tree above shows a possible implementation of a tree for registering web authentication devices.

After verifying the users credentials against the configured data store, tree evaluation continues to the WebAuthn Registration Node.

If the user's client does not support web authentication, the failure URL is altered, for example to redirect the user to a page explaining which clients and operating systems support web authentication.

If the user's client does support web authentication, and the connection is secured with TLS, the user will be asked to register an authenticator:

The WebAuthn Registration node waiting for an authenticator.



Waiting for local device...

info@forgerock.com
Copyright © 2010-2018 ForgeRock AS. All rights reserved.

The user's browser may present a consent pop-up to allow access to the authenticators available on the client. When consent has been granted the browser activates the relevant authenticators, ready for registration.

Tip

The relying party details configured in the node are often included in the consent message to help the user verify the entity that is requesting access.

The authenticators the client activates for registration depends in the value of the properties in the node. For example, if the `User verification requirement` property is set to `REQUIRED`, the client would not activate a USB hardware security key for registration.

When the user completes an authorization gesture, for example scanning a fingerprint, or entering a PIN number, tree evaluation continues along the `Success` outcome path, and in this example will be taken to their profile page.

The registered authenticator appears on the user's dashboard page, with the label *New Security Key*. To rename the authenticator, click its vertical ellipsis context icon (⋮), and then click Rename.

Any problems encountered during the registration, including a timeout, results in tree evaluation continuing to the `Failure` outcome.

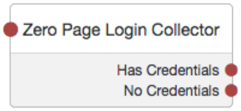
Zero Page Login Collector Node

The Zero Page Login Collector authentication node checks whether selected headers are provided in the incoming authentication request, and if so, uses their value as the provided username and password.

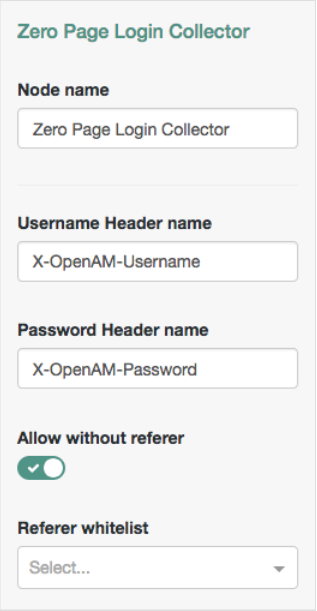
The tree evaluation continues along the `Has Credentials` outcome path if the specified headers are available in the request, or the `No Credentials` path if the specified headers are not present.

A common use for the Zero Page Login Collector authentication node is to connect the `Has Credentials` outcome connector to the input of a Data Store Decision node, and the `No Credentials` outcome connector to the input of a Username Collector node followed by a Password Collector node, and then into the same Data Store Decision node as earlier. For an example of this layout, see the default `Example` authentication tree provided in AM. See "Example Tree With Zero Page Login Node".

The password collected by the Zero Page Login Collector node is transient, persisting only until the authentication flow reaches the next node requiring user interaction.



Properties:



Zero Page Login Collector

Node name

Zero Page Login Collector

Username Header name

X-OpenAM-Username

Password Header name

X-OpenAM-Password

Allow without referer

✓

Referer whitelist

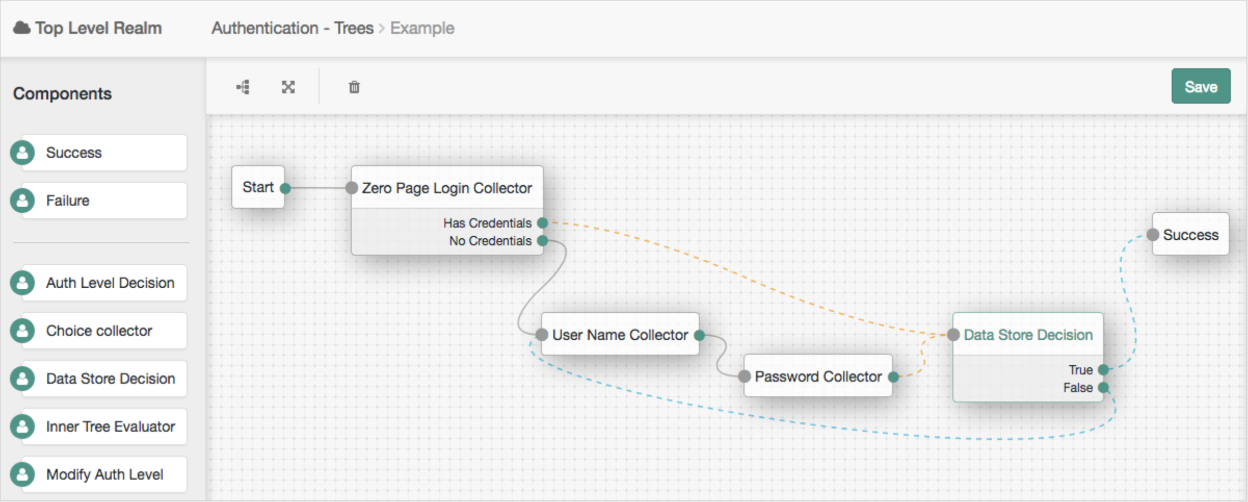
Select...

Property	Usage
Username Header name	Enter the name of the header that contains the username value.

Property	Usage
	Default: `X-OpenAM-Username`
Password Header name	Enter the name of the header that contains the password value. Default: `X-OpenAM-Password`
Allow without referer	If enabled, the node accepts incoming requests that do not contain a `Referer` HTTP header. If a `Referer` HTTP header is present, the value is not checked. If disabled, a `Referer` HTTP header must be present in the incoming request, and the value must appear in the Referer whitelist property. Default: `Enabled`
Referer whitelist	Specify a list of URLs allowed in the `Referer` HTTP header of incoming requests. Incoming requests containing a `Referer` HTTP header value not specified in the whitelist causes tree evaluation to continue along the `No Credentials` outcome path. **Note** You must disable the Allow without referer property for the Referer whitelist property to take effect.

Example:

Example Tree With Zero Page Login Node



Configuring Authentication Webhooks

This section covers creating webhooks, which are used to send HTTP POST calls to a server with contextual information about an authentication session when a predefined event occurs, for example, logging out.

Webhooks are used from within authentication trees, by the following nodes:

- Register Logout Webhook Node

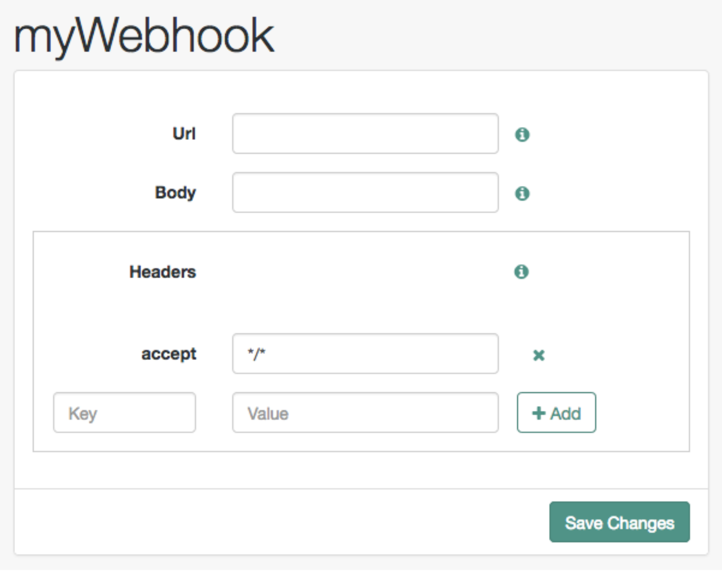
To Create an Authentication Webhook

Perform the following steps to create an authentication webhook for use within an authentication tree:

1. Log in to the AM console as an administrator, for example, `amadmin`.
2. Navigate to Realms > *Realm Name* > Authentication > Webhooks.

   - To create a new webhook, select Create Webhook, specify a Webhook Name, and then select Create.
   - To edit an existing webhook, select the name of the webhook.

   A screen similar to the following appears:



myWebhook

Url	
Body	
Headers	
accept	*/*

3. Complete the fields as required:

Url

Specifies the URL to which the HTTP POST is sent when the event occurs.

Body

Specifies the body of the HTTP POST. You can send different formats by also setting the correct Content-Type header in the `Header` property, for example:

- **Form Data**. Enter the body value in the format `parameter=value&parameter2=value2`, and set a `Content-Type` header of `application/x-www-form-urlencoded`.
- **JSON Data**. Enter the body value in the format `{"parameter":"value","parameter2":"value2"}`, and set a `Content-Type` header of `application/json`.

Headers

Specifies any HTTP headers to add to the POST.

To add a header, enter the name of the header in the `Key` field, and the value, and then click the Add button (✚).

To remove a header, select the Delete button (✖).

Each of the fields in a webhook supports variables for retrieving values from the user's session after successfully authenticating. Specify a variable in the following format:

```
${variable_name}
```

Any custom properties added to the session using the Set Session Properties Node can be accessed by using a variable, as well as the following session properties:

`AMCtxId`

`amlbcookie`

`authInstant`

`AuthLevel`

`CharSet`

`clientType`

`FullLoginURL`

`Host`

`HostName`

`IndexType`

`Locale`

`Organization`

`Principal`

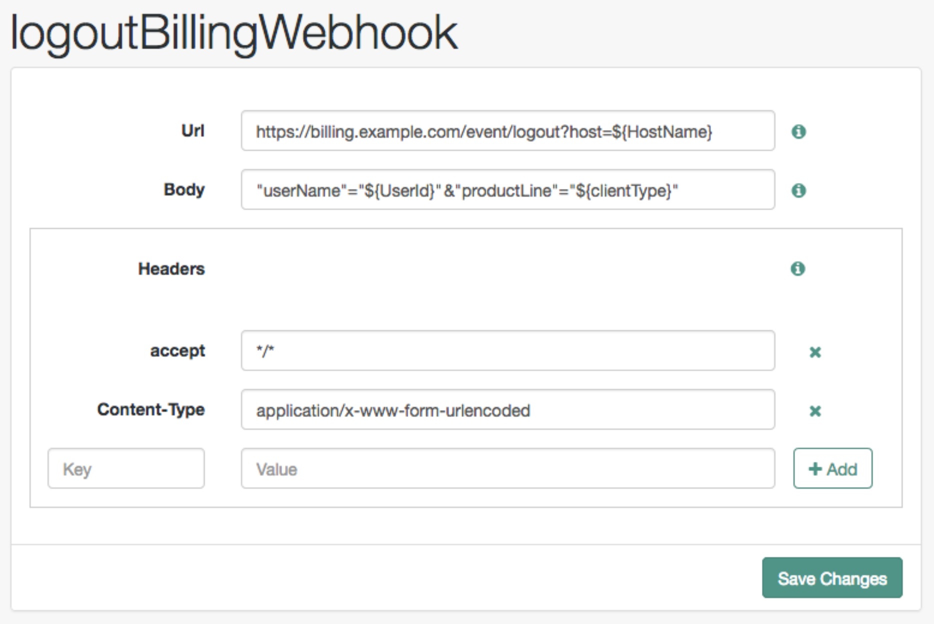
`Principals`

`Service`

`successURL`

`sun.am.UniversalIdentifier`
`UserId`
`UserProfile`
`UserToken`
`webhooks`

The following figure shows an example webhook, using variable substitutions:



logoutBillingWebhook

Url: https://billing.example.com/event/logout?host=${HostName}

Body: "userName"="${UserId}"&"productLine"="${clientType}"

Headers

accept: */*

Content-Type: application/x-www-form-urlencoded

> **Warning**
>
> Specifying a variable that is not present in the user's session places the literal variable text in to the HTTP POST, for example `user=${UserId}`, rather than `user=demo`.

Configuring Authentication Chains and Modules

AM provides several authentication modules and chains, the most important being the following:

- The `Amster` authentication module and `amsterService` authentication chain, used by Amster clients.
- The `DataStore` authentication module and `ldapService` authentication chain, configured for all users in new installations.

To configure authentication modules and chains, see the following sections:

- "Configuring Authentication Modules"
- "Configuring Authentication Chains"

Configuring Authentication Modules

The AM console provides two places where you can configure authentication modules:

1. Under Configure > Authentication, you configure default properties for global authentication modules.

2. Under Realms > *Realm Name* > Authentication > Modules, you configure modules for your realm.

The configuration of individual modules depend on its function. The configuration of an Active Directory instead of the LDAP authentication module requires connection information and details about where to search for users. In contrast, the configuration of the HOTP module for OTP authentication requires data about the password length and the mail server or SMS gateway to send the password during authentication.

Active Directory Authentication Module

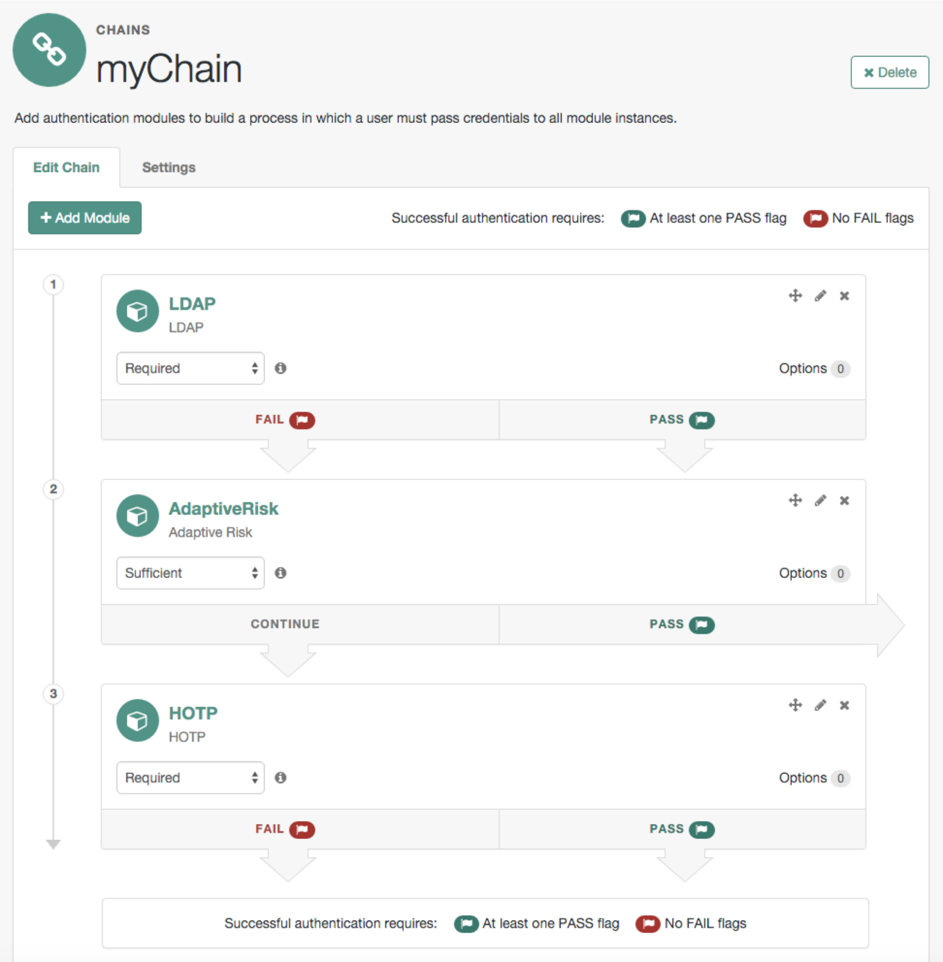
AM connects to Active Directory over Lightweight Directory Access Protocol (LDAP). AM provides separate Active Directory and LDAP modules to support the use of both Active Directory and another directory service in an authentication chain.

For detailed information about this module's configuration properties, see "Active Directory Module Properties".

Adaptive Risk Authentication Module

The Adaptive Risk module is designed to assess risk during authentication, so that AM can determine whether to require the user to complete further authentication steps. After configuring the Adaptive Risk module, insert it in your authentication chain with criteria set to Sufficient as shown in the following example:

Adaptive Risk Module in an Authentication Chain



CHAINS

myChain

✖ Delete

Add authentication modules to build a process in which a user must pass credentials to all module instances.

Edit Chain | Settings

+ Add Module

Successful authentication requires: At least one PASS flag, No FAIL flags

1 LDAP (LDAP): Required. Options 0. FAIL / PASS

2 AdaptiveRisk (Adaptive Risk): Sufficient. Options 0. CONTINUE / PASS

3 HOTP (HOTP): Required. Options 0. FAIL / PASS

Successful authentication requires: At least one PASS flag, No FAIL flags

In the example authentication chain shown, AM has users authenticate first using the LDAP module providing a user ID and password combination. Upon success, AM calls the Adaptive Risk module. The Adaptive Risk module assesses the risk based on your configured parameters. If the Adaptive Risk module calculates a total score below the threshold you set, the module returns success, and AM finishes authentication processing without requiring further credentials. Otherwise, the Adaptive Risk module evaluates the score to be above the risk threshold, and returns failure. AM then calls the HOTP module, requiring the user to authenticate with a one-time password delivered to her by email or by SMS to her mobile phone.

When you configure the Adaptive Risk module to save cookies and profile attributes after successful authentication, AM performs the save as post-authentication processing, only after the entire authentication chain returns success. You must set up AM to save the data as part of post-authentication processing by editing the authentication chain to add `org.forgerock.openam.authentication.modules.adaptive.AdaptivePostAuthenticationPlugin` to the list of post-authentication plugins.

When the Adaptive Risk module relies on the client IP address, and AM lies behind a load balancer or proxy layer, configure the load balancer or proxy to send the address by using the `X-Forwarded-For` header, and configure AM to consume and forward the header as necessary. For details, see "Handling HTTP Request Headers" in the *Installation Guide*.

For detailed information about this module's configuration properties, see "Adaptive Risk Authentication Module Properties".

Anonymous Authentication Module

This module lets you configure and track anonymous users, who can log in to your application or web site without login credentials. Typically, you would provide such users with very limited access. For example, an anonymous user may have access to public downloads on your site. When the user attempts to access resources that require more protection, the module can force further authentication for those resources.

The anonymous user is enabled by default. To harden security, deactivate the anonymous user, unless anonymous access is specifically required in your deployment. See How do I deactivate the default anonymous user in AM.

For detailed information about this module's configuration properties, see "Anonymous Authentication Module Properties".

Certificate Authentication Module

X.509 digital certificates can enable secure authentication without the need for user names and passwords or other credentials. Certificate authentication can be used to manage authentication by applications. If all certificates are signed by a recognized Certificate Authority (CA), then you might not need additional configuration. If you need to look up public keys of AM clients, this module can also look up public keys in an LDAP directory server.

When you store certificates and certificate revocation lists (CRL) in an LDAP directory service, you must configure:

- How to access the directory service.
- How to look up the certificates and CRLs, based on the fields in the certificates that AM clients present to authenticate.

Access to the LDAP server and how to search for users is similar to LDAP module configuration as in "LDAP Authentication Module". The primary difference is that, unlike for LDAP configuration, AM retrieves the user identifier from a field in the certificate that the client application presents, then uses that identifier to search for the LDAP directory entry that holds the certificate, which should match the certificate presented. For example, if the Subject field of a typical certificate has a DN `C=FR, O=Example Corp, CN=Barbara Jensen`, and Barbara Jensen's entry in the directory has `cn=Barbara Jensen`, then you can use `CN=Barbara Jensen` from the Subject DN to search for the entry with `cn=Barbara Jensen` in the directory.

For detailed information about this module's configuration properties, see "Certificate Authentication Module Properties".

Data Store Authentication Module

The Data Store authentication module allows a login using the identity repository of the realm to authenticate users. The Data Store module removes the requirement to write an authentication plugin module, load, and then configure the authentication module if you need to authenticate against the same data store repository. Additionally, you do not need to write a custom authentication module where flatfile authentication is needed for the corresponding repository in that realm.

The Data Store module is generic. It does not implement data store-specific capabilities, such as the password policy and password reset features provided by LDAP modules. Therefore, the Data Store module returns failure when such capabilities are invoked.

For detailed information about this module's configuration properties, see "Data Store Authentication Module Properties".

Device ID (Match) Authentication Module

The Device ID (Match) module provides device fingerprinting functionality for risk-based authentication. The Device ID (Match) module collects the unique characteristics of a remote user's computing device and compares them to characteristics on a saved device profile. The module computes any variances between the collected characteristics to those stored on the saved device profile and assigns penalty points for each difference.

For detailed information about this module's configuration properties, see "Device ID (Match) Authentication Module Properties".

In general, you can configure and gather the following device characteristics:

- User agents associated with the configuration of a web browser
- Installed fonts
- Plugins installed for the web browser
- Resolution and color depth associated with a display
- Timezone or geolocation of a device

For example, when a user who typically authenticates to AM using Firefox and then logs on using Chrome, the Device ID (Match) module notes the difference and assigns penalty points to this change in behavior. If the module detects additional differences in behavior, such as browser fonts, geolocation, and so forth, then additional points are assessed and calculated.

If the total number of penalty points exceeds a pre-configured threshold value, the Device ID (Match) module fails and control is determined by how you configured your authentication chain. If you include the HOTP module in your authentication chain, and if the Device ID (Match) module fails after the maximum number of penalty points have been exceeded, then the authentication chain issues a HOTP request to the user, requiring the user to identify themselves using two-factor authentication.

Important

By default, the maximum penalty points is set to 0, which you can adjust in the server-side script.

The Device ID (Match) module comes pre-configured with default client-side and server-side JavaScript code, supplying the logic necessary to fingerprint the user agent and computer. Scripting allows you to customize the code, providing more control over the device fingerprint elements that you would like to collect. While AM scripting supports both the JavaScript (default) and Groovy languages, only server-side scripts can be written in either language. The client-side scripts must be written in the JavaScript language.

Caution

The Device ID (Match) module's default JavaScript client-side and server-side scripts are fully functional. If you change the client-side script, you must also make a corresponding change to the server-side script. For a safer option, if you want to change the behavior of the module, you can make a copy of the scripts, customize the behavior, and update the Device ID (Match) modules to use the new scripts.

The Device ID (Match) module does not stand on its own within an authentication chain and requires additional modules. For example, you can have any module that identifies the user (for example, DataStore, Active Directory or others), Device ID (Match), any module that provides two-factor authentication, for example the ForgeRock Authenticator (OATH) or ForgeRock Authenticator (Push) authentication modules, and Device ID (Save) within your authentication chain.

As an example, you can configure the following modules with the specified criteria:

1. **DataStore - Requisite**. The Device ID (Match) module requires user authentication information to validate the username. You can also use other modules that identify the username, such as LDAP, Active Directory, or RADIUS.

2. **Device ID (Match) - Sufficient**. The Device ID (Match) runs the client-side script, which invokes the device fingerprint collectors, captures the data, and converts it into a JSON string. It then auto-submits the data in a JSP page to the server-side scripting engine.

   The server-side script calculates the penalty points based on differences between the client device and stored device profile, and whether the client device successfully "matches" the stored profile.

If a match is successful, AM determines that the client's device has the required attributes for a successful authentication.

If the device does not have a match, then the module fails and falls through to the HOTP module for further processing.

3. **HOTP - Requisite**. If the user's device does not match a stored profile, AM presents the user with a HMAC One-Time Password (HOTP) screen either by SMS or email, prompting the user to enter a password.

   You can also use any other module that provides two-factor authentication.

   After the HOTP has successfully validated the user, the Device ID (Save) module gathers additional data from the user. For specific information about the HOTP module, see "HOTP Authentication Module".

4. **Device ID (Save) - Required**. The Device ID (Save) module provides configuration options to enable an auto-save feature on the device profile as well as set a maximum number of stored device profiles on the user entry or record. Once the maximum number of stored device profiles is reached, AM deletes the old data from the user record as new ones are added. User records could thus contain both old and new device profiles.

   If the auto-save feature is not enabled, AM presents the user with a screen to save the new device profile.

   The module also takes the device print and creates a JSON object that includes the ID, name, last selected date, selection counter, and device print. For specific information about the Device ID (Save) module, see "Device ID (Save) Module".

   > **Note**
   >
   > If a user has multiple device profiles, the profile that is the closest match to the current client details is used for the comparison result.

To Configure the Device ID (Match) Authentication Module

1. Log into the AM console as an administrator.

2. On the Realms page, click the realm from which you want to work.

3. Click Authentication > Modules.

4. To add the Device ID (Match) module, do the following substeps:

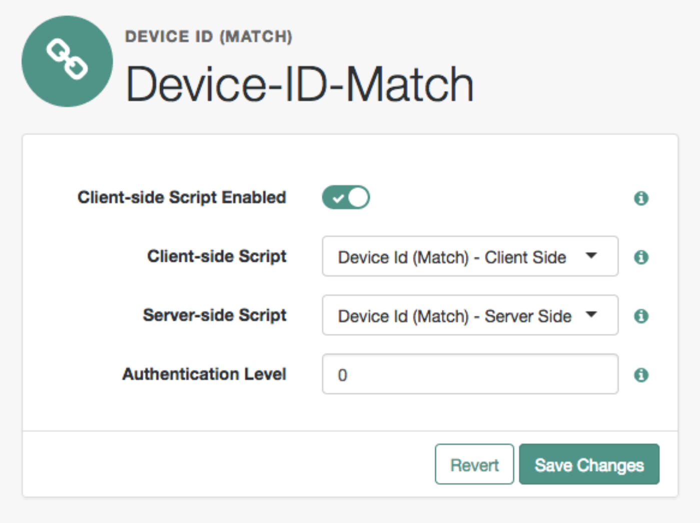
   a. Click Add Module.

   b. In the Module Name box, enter `Device-ID-Match`.

c. In the Type box, select `Device Id (Match)`, and then click Create.

d. Click Save Changes.

Device ID (Match) Module



DEVICE ID (MATCH)

Device-ID-Match

Client-side Script Enabled: ☑

Client-side Script: Device Id (Match) - Client Side

Server-side Script: Device Id (Match) - Server Side

Authentication Level: 0

5. To make adjustments to the default scripts, click Scripts drop-down list, and then click `Device Id (Match) - Client Side`.
6. To make corresponding changes to the server-side script, click Scripts drop-down list, and then click `Device Id (Match) - Server Side`. For more information, see "Managing Scripts".

To Configure an Authentication Chain With a Device ID (Match) Authentication Module

1. Log into the AM console as an administrator.
2. On the Realms page, click the realm from which you want to work.
3. Click Authentication > Chains.
4. On the Authentication Chains page, do the following steps:

   a. Click Add Chain. In the Chain Name box, enter a descriptive label for your authentication chain, and then click Create.

   b. Click Add Module.

   c. On the New Module dialog, select the authentication module, select the criteria, and then click Ok to save your changes. Repeat the last two steps to enter each module to your chain.

For example, you can enter the following modules and criteria:

Device ID Chain

Module	Criteria
DataStore	REQUISITE
Device-ID-Match	SUFFICIENT
HOTP	REQUISITE
Device-ID-Save	REQUIRED

It is assumed that you have added the Device Id (Match) and Device Id (Save) modules. If you have not added these modules, see "To Configure the Device ID (Match) Authentication Module" and "To Configure the Device ID (Save) Authentication Module".

5. Review your authentication chain, and then click Save Changes.

What the User Sees During Authentication

When the user logs on to the AM console, AM determines if the user's device differs from that of the stored profile. If the differences exceed the maximum number of penalty points or a device profile has not yet been stored, AM sends an "Enter OTP" page, requiring the user to enter a one-time password, which is sent to the user via email or SMS. The user also has the option to request a one-time password.

Next, because the Device ID (Save) module is present, AM presents the user with a "Add to Trusted Devices?" page, asking if the user wants to add the device to the list of trusted device profiles. If the user clicks "Yes", AM prompts the user to enter a descriptive name for the trusted device.

Next, AM presents the user with the User Profile page, where the user can click the Dashboard link at top to access the My Applications and Authentication Devices page. Once on the Dashboard, the user can view the list of trusted devices or remove the device by clicking the Delete Device link.

Device ID (Save) Module

The Device ID (Save) module saves a user's device profile. The module can either save the profile upon request, requiring the user to provide a name for the device and explicitly save it, or it can save the profile automatically. If a user has multiple device profiles, the profile that is the closest match to the current client details is used for the comparison result.

For detailed information about this module's configuration properties, see "Device ID (Save) Authentication Module Properties".

Within its configured authentication chain, the Device ID (Save) module also takes the device print and creates a JSON object that consists of the ID, name, last selected date, selection counter, and device print itself.

To Configure the Device ID (Save) Authentication Module

1. Log into the AM console as an administrator.

2. Click the realm from which you want to work.

3. Click Authentication > Modules.

4. To add the Device ID (Save) module, click Add Module.

5. In the Module Name box, enter `Device-ID-Save`.

6. In the Type box, select `Device Id (Save)`, and then click Create.

7. To configure the Device-Id (Save) module, do the following:

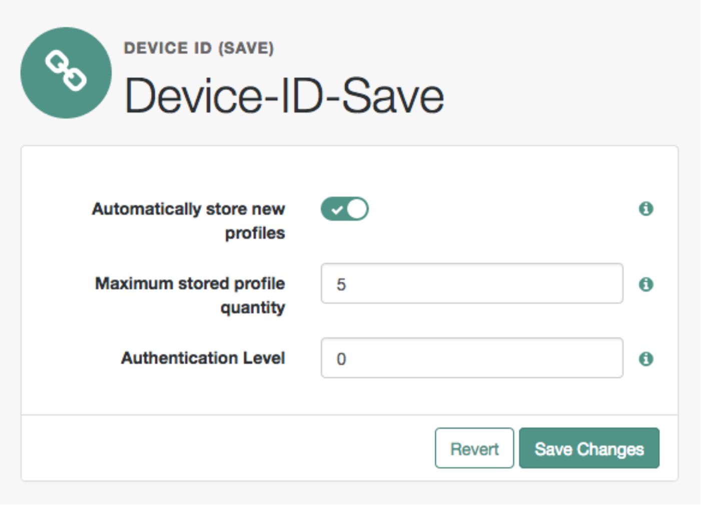
   a. Click the Automatically store new profiles checkbox. If this box is left unchecked, the user will be prompted to give consent to store new profiles.

   b. In the Maximum stored profile quantity box, enter the max number of stored profiles. Any profile that exceeds this number will not be stored.

   c. In the Authentication Level box, enter a number corresponding to the authentication level of the module.

   d. Click Save Changes.

Device ID (Save) Module



DEVICE ID (SAVE)

Device-ID-Save

Automatically store new profiles: ✓ (on)

Maximum stored profile quantity: 5

Authentication Level: 0

Federation Authentication Module

The Federation authentication module is used by a service provider to create a user session after validating single sign-on protocol messages. This authentication module is used by the SAML, SAMLv2, ID-FF, and WS-Federation protocols.

For detailed information about this module's configuration properties, see "Federation Authentication Module Properties".

ForgeRock Authenticator (OATH) Authentication Module

The ForgeRock Authenticator (OATH) module provides a more secure method for users to access their accounts with the help of a device such as a mobile phone. For detailed information about two-step verification with the ForgeRock Authenticator (OATH) module in AM, see "*Implementing Multi-Factor Authentication*".

For detailed information about this module's configuration properties, see "ForgeRock Authenticator (OATH) Authentication Module Properties".

Note

AM provides two authentication modules that support OATH:

- The ForgeRock Authenticator (OATH) authentication module, which is optimized for use with the ForgeRock Authenticator app and provides device profile encryption.
- The OATH authentication module, which is a raw OATH implementation requiring more configuration for users and the AM administrator.

We recommend using the ForgeRock Authenticator (OATH) authentication module when possible.

Also, the ForgeRock Authenticator (OATH), HOTP, and OATH authentication modules all support HOTP passwords, but the way that users obtain passwords differs. See "Differences Among Authentication Modules That Support HOTP" for more information.

ForgeRock Authenticator (Push) Authentication Module

The ForgeRock Authenticator (Push) module provides a way to send push notification messages to a device such as a mobile phone, enabling multi-factor authentication. For detailed information about multi-factor authentication with the ForgeRock Authenticator (Push) module in AM, see "*Implementing Multi-Factor Authentication*".

For detailed information about this module's configuration properties, see "ForgeRock Authenticator (Push) Authentication Module Properties".

ForgeRock Authenticator (Push) Registration Authentication Module

The ForgeRock Authenticator (Push) Registration module provides a way to register a device such as a mobile phone for multi-factor authentication. For detailed information about multi-factor authentication with the ForgeRock Authenticator (Push) module in AM, see "Managing Devices for Multi-Factor Authentication".

For detailed information about this module's configuration properties, see "ForgeRock Authenticator (Push) Registration Authentication Module Properties".

HOTP Authentication Module

The HOTP authentication module works with an authentication chain with any module that stores the `username` attribute. The module uses the `username` from the `sharedState` set by the previous module in the chain and retrieves the user's email address or telephone number to send a one-time password to the user. The user then enters the password on a Login page and completes the authentication process if successful.

For example, to set up HOTP in an authentication chain, you can configure the Data Store module (or any module that stores the user's `username`) as the `requisite` first module, and the HOTP module as the second `requisite` module. When authentication succeeds against the Data Store module, the HOTP module retrieves the Email Address and Telephone Number attributes from the data store based on the `username` value. For the HOTP module to use either attribute, the Email Address must contain a valid email address, or the Telephone Number must contain a valid SMS telephone number.

You can set the HOTP module to automatically generate a password when users begin logging into the system. You can also set up mobile phone, mobile carrier, and email attributes for tighter controls over where the messages are generated and what provider the messages go through to reach the user.

For detailed information about this module's configuration properties, see "HOTP Authentication Module Properties".

Note

The ForgeRock Authenticator (OATH), HOTP, and OATH authentication modules all support HOTP passwords, but the way that users obtain passwords differs. See "Differences Among Authentication Modules That Support HOTP" for more information.

HTTP Basic Authentication Module

HTTP basic authentication takes a user name and password from HTTP authentication and tries authentication against the backend module in AM, depending on what you configure as the Backend Module Name.

For detailed information about this module's configuration properties, see "HTTP Basic Authentication Module Properties".

JDBC Authentication Module

The Java Database Connectivity (JDBC) module lets AM connect to a database, such as MySQL or Oracle DB to authenticate users.

For detailed information about this module's configuration properties, see "JDBC Authentication Module Properties".

LDAP Authentication Module

AM connects to directory servers using Lightweight Directory Access Protocol (LDAP). To build an easy-to-manage, high-performance, pure Java directory service, try ForgeRock Directory Services.

For detailed information about this module's configuration properties, see "LDAP Authentication Module Properties".

Legacy OAuth 2.0/OpenID Connect Authentication Module

Note

Use of this authentication module is deprecated. Use the replacements instead, as described in "Social Authentication Modules"

The Legacy OAuth 2.0/OpenID Connect authentication module lets AM authenticate clients of OAuth resource servers. References in this section are to RFC 6749, The OAuth 2.0 Authorization Framework.

If the module is configured to create an account if none exists, then you must provide valid SMTP settings. As part of account creation, the OAuth 2.0/OpenID Connect client authentication module sends the resource owner an email with an account activation code. To send email, AM uses the SMTP settings from the configuration for the OAuth 2.0/OpenID Connect authentication module.

For detailed information about this module's configuration properties, see "Legacy OAuth 2.0/OpenID Connect Authentication Module Properties".

MSISDN Authentication Module

The Mobile Station Integrated Services Digital Network (MSISDN) authentication module enables non-interactive authentication using a mobile subscriber ISDN associated with a terminal, such as a mobile phone. The module checks the subscriber ISDN against the value found on a user's entry in an LDAP directory service.

For detailed information about this module's configuration properties, see "MSISDN Authentication Module Properties".

OATH Authentication Module

The Open Authentication (OATH) module provides a more secure method for users to access their accounts with the help of a device, such as their mobile phone or Yubikey. Users can log into AM and update their information more securely from a one-time password (OTP) displayed on their device.

The OATH module includes the OATH standard protocols (RFC 4226 and RFC 6238). The OATH module has several enhancements to the HMAC One-Time Password (HOTP) Authentication Module, but does not replace the original module for those already using HOTP prior to the 10.1.0 release. The OATH module includes HOTP authentication and Time-Based One-Time Password (TOTP) authentication. Both types of authentication require an OATH compliant device that can provide the OTP.

HOTP authentication generates the OTP every time the user requests a new OTP on their device. The device tracks the number of times the user requests a new OTP, called the counter. The OTP displays for a period of time you designate in the setup, so the user may be further in the counter on their device than on their account. AM will resynchronize the counter when the user finally logs in. To accommodate this, you set the number of passwords a user can generate before their device cannot be resynchronized. For example, if you set the number of HOTP Window Size to 50 and someone presses the button 30 on the user's device to generate a new OTP, the counter in AM will review the OTPs until it reaches the OTP entered by the user. If someone presses the button 51 times, you will need to reset the counter to match the number on the device's counter before the user can login to AM. HOTP authentication does not check earlier passwords, so if the user attempts to reset the counter on their device, they will not be able to login until you reset the counter in AM to match their device. See "Resetting Registered Devices by using REST" for more information.

TOTP authentication constantly generates a new OTP based on a time interval you specify. The device tracks the last two passwords generated and the current password. The Last Login Time monitors the time when a user logs in to make sure that user is not logged in several times within the present time period. Once a user logs into AM, they must wait for the time it takes TOTP to generate the next two passwords and display them. This prevents others from being able to access the users account using the OTP they entered. The user's account can be accessed again after the generation of the third new OTP is generated and displayed on their device. For this reason, the TOTP Time-Step Interval should not be so long as to lock users out, with a recommended time of 30 seconds.

An authentication chain can be created to generate an OTP from either HOTP or TOTP.

For detailed information about this module's configuration properties, see "OATH Authentication Module Properties".

Note

AM provides two authentication modules that support OATH:

- The ForgeRock Authenticator (OATH) authentication module, which is optimized for use with the ForgeRock Authenticator app and provides device profile encryption.
- The OATH authentication module, which is a raw OATH implementation requiring more configuration for users and the AM administrator.

We recommend using the ForgeRock Authenticator (OATH) authentication module when possible.

Also, the ForgeRock Authenticator (OATH), HOTP, and OATH authentication modules all support HOTP passwords, but the way that users obtain passwords differs. See "Differences Among Authentication Modules That Support HOTP" for more information.

OpenID Connect id_token bearer Module

The OpenID Connect id_token bearer module lets AM rely on an OpenID Connect 1.0 provider's ID Token to authenticate an end user.

Note

This module validates an OpenID Connect ID token and matches it with a user profile. You should not use this module if you want AM to act as a client in the full OpenID Connect authentication flow.

To provision AM as an OpenID Connect client, you should instead configure an OAuth 2.0 or OpenID Connect social auth module. AM also provides a wizard to configure an OpenID Connect module that will authenticate against an OpenID Connect 1.0 provider. For more information, see "Configuring Custom Social Authentication Providers".

The OpenID Connect id_token bearer module expects an OpenID Connect ID Token in an HTTP request header. It validates the ID Token, and if successful, looks up the AM user profile corresponding to the end user for whom the ID Token was issued. Assuming the ID Token is valid and the profile is found, the module authenticates the AM user.

You configure the OpenID Connect id_token bearer module to specify how AM gets the information needed to validate the ID Token, which request header contains the ID Token, the issuer identifier for the provider who issued the ID Token, and how to map the ID Token claims to an AM user profile.

OpenID Connect id_token Bearer Example

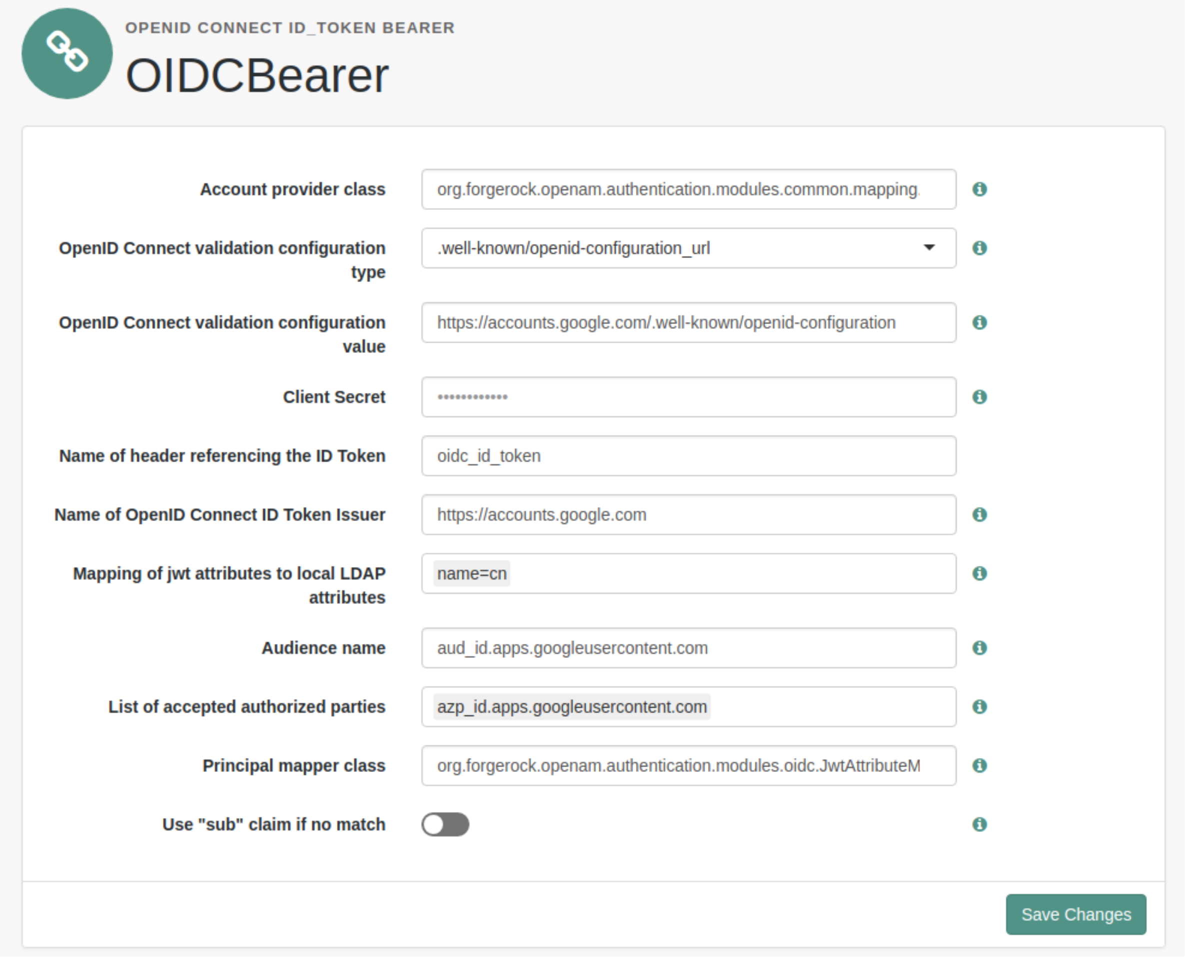
The OpenID Connect id_token bearer module configuration must match the claims returned in the `id_token` JWT used to authenticate.

Before configuring the module, use an OpenID Connect client to obtain an `id_token`. Decode the `id_token` value to see the claims in the middle portion of the JWT. The claims in the decoded `id_token` look something like the following example, which was obtained at Google's OAuth 2.0 Playground:

```
{
      azp: "azp_id.apps.googleusercontent.com",
      aud: "aud_id.apps.googleusercontent.com",
      sub: "subject_id",
      at_hash: "access_token_hash",
      iss: "https://accounts.google.com",
      iat: 1505814261,
      exp: 1505817861,
      name: "Google User",
      picture: "https://lh5.googleusercontent.com/***/photo.jpg",
      given_name: "Google",
      family_name: "User",
      locale: "en"
      }
```

The azp, aud, and iss values are literally reused in the module configuration. Also notice that, in this example, name is mapped to cn for user accounts in the identity repository. The following figure shows an example configuration for this id_token format.

Sample OpenID Connect id_token Bearer Module Configuration



OPENID CONNECT ID_TOKEN BEARER

OIDCBearer

Account provider class	org.forgerock.openam.authentication.modules.common.mapping.
OpenID Connect validation configuration type	.well-known/openid-configuration_url
OpenID Connect validation configuration value	https://accounts.google.com/.well-known/openid-configuration
Client Secret	•••••••••••••
Name of header referencing the ID Token	oidc_id_token
Name of OpenID Connect ID Token Issuer	https://accounts.google.com
Mapping of jwt attributes to local LDAP attributes	name=cn
Audience name	aud_id.apps.googleusercontent.com
List of accepted authorized parties	azp_id.apps.googleusercontent.com
Principal mapper class	org.forgerock.openam.authentication.modules.oidc.JwtAttributeM
Use "sub" claim if no match	○

The following example command demonstrates a REST call that authenticates the user using the module:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--header "oidc_id_token: eyJ...ifQ.eyJ...In0.BT1...iZA" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate?
authIndexType=module&authIndexValue=OIDCBearer'
{
    "tokenId": "nIq...AA*",
    "successUrl": "/openam/console",
    "realm": "/"
}
```

Notice that the `id_token` value, abbreviated as `eyJ...ifQ.eyJ...In0.BT1...iZA`, is the value of the `oidc_id_token` header as seen in the configuration. The request targets a module named `OIDCBearer` as specified by the `authIndexType` and `authIndexValue` parameters. For detailed information about the authentication REST API, see "Authentication and Logout using REST".

For detailed information about this module's configuration properties, see "OpenID Connect id_token bearer Authentication Module Properties".

Persistent Cookie Module

The Persistent Cookie module supports the configuration of cookie lifetimes based on requests and a maximum time. Note that by default, the persistent cookie is called `session-jwt`.

Important

If Secure Cookie is enabled (Deployment > Servers > *Server Name* > Security > Cookie), the Persistent Cookie module only works over HTTPS.

The module signs and encrypts the JSON Web Token (JWT) that is inserted as the value of the persistent cookie. The relevant secret IDs and the default public key and HMAC key aliases are shown in the table below:

Secret ID Mappings for Persistent Cookies

The following table shows the default secret ID mappings used to encrypt and then sign persistent cookies:

Secret ID	Default Alias	
am.default.authentication.modules.persistentcookie.encryption	test	
am.default.authentication.modules.persistentcookie.signing	hmacsigningtest	

For each instance of a persistent cookie module available in a realm, there is a dynamic secret ID associated with that module configuration instance.

For example, in a single realm you can have a Persistent Cookie module instance with the name *helloworld*, and a separate Persistent Cookie module instance with the name *hellomars*.

The following secret ID mappings could be used to encrypt and then sign persistent cookies:

Secret ID	Default Alias	
am.authentication.modules.persistentcookie.*helloworld*.encryption	helloworld	
am.authentication.modules.persistentcookie.*helloworld*.signing	hmacsigninghelloworld	
am.authentication.modules.persistentcookie.*hellomars*.encryption	hellomars	
am.authentication.modules.persistentcookie.*hellomars*.signing	hmacsigninghellomars	

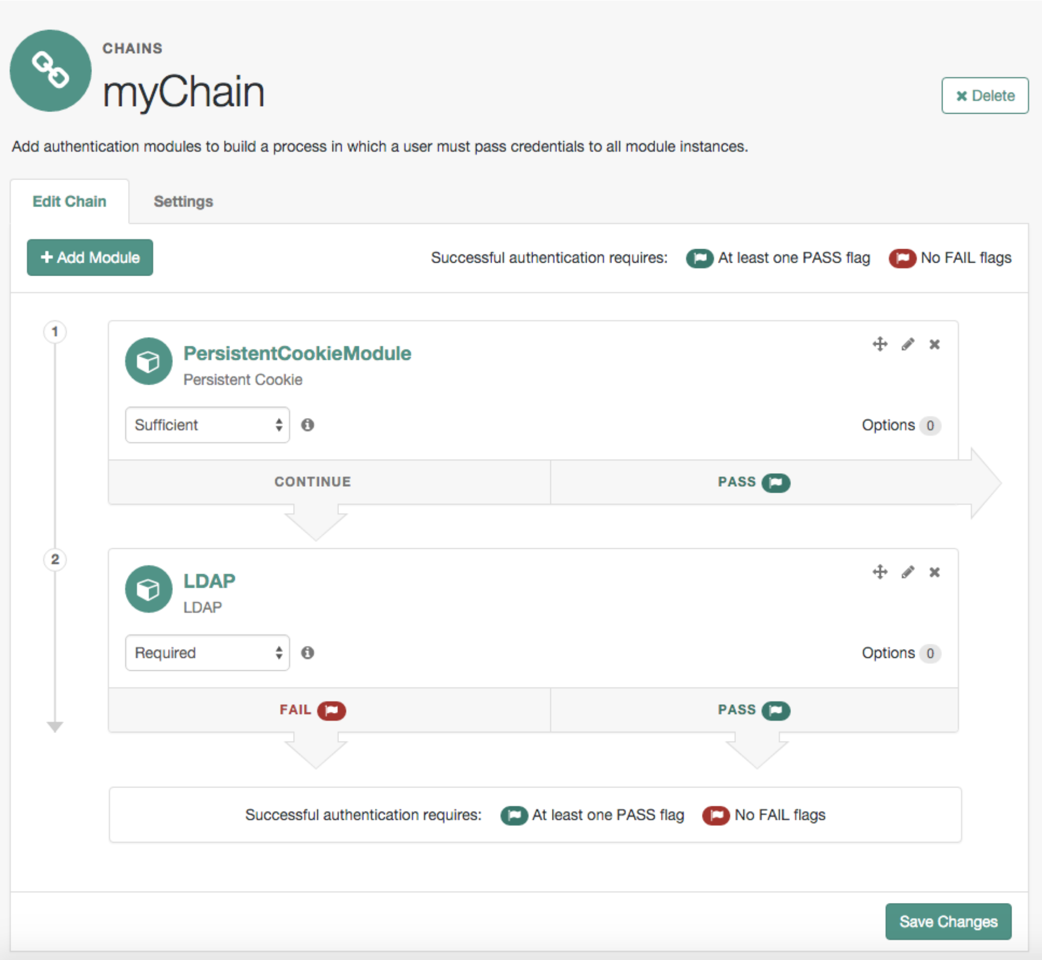
AM will attempt to look up the secrets with the Persistent Cookie module instance name. If unsuccessful, AM will look up the secrets using the default secret ID.

For information on mapping certificate aliases to secret IDs in secret stores, see "Mapping Secrets" in the *Setup and Maintenance Guide*.

When the Persistent Cookie module enforces the client IP address, and AM lies behind a load balancer or proxy layer, configure the load balancer or proxy to send the address by using the `X-Forwarded-For` header, and configure AM to consume and forward the header as necessary. For details, see "Handling HTTP Request Headers" in the *Installation Guide*.

The Persistent Cookie module belongs with a second module in an authentication chain. To see how this works, navigate to Realms > *Realm Name* > Authentication > Chains. Create a new chain and add modules as shown in the figure. The following example shows how a Persistent Cookie module is sufficient. If the persistent cookie does not yet exist, authentication relies on LDAP:

Persistent Cookie Module in an Authentication Chain



CHAINS

myChain

Add authentication modules to build a process in which a user must pass credentials to all module instances.

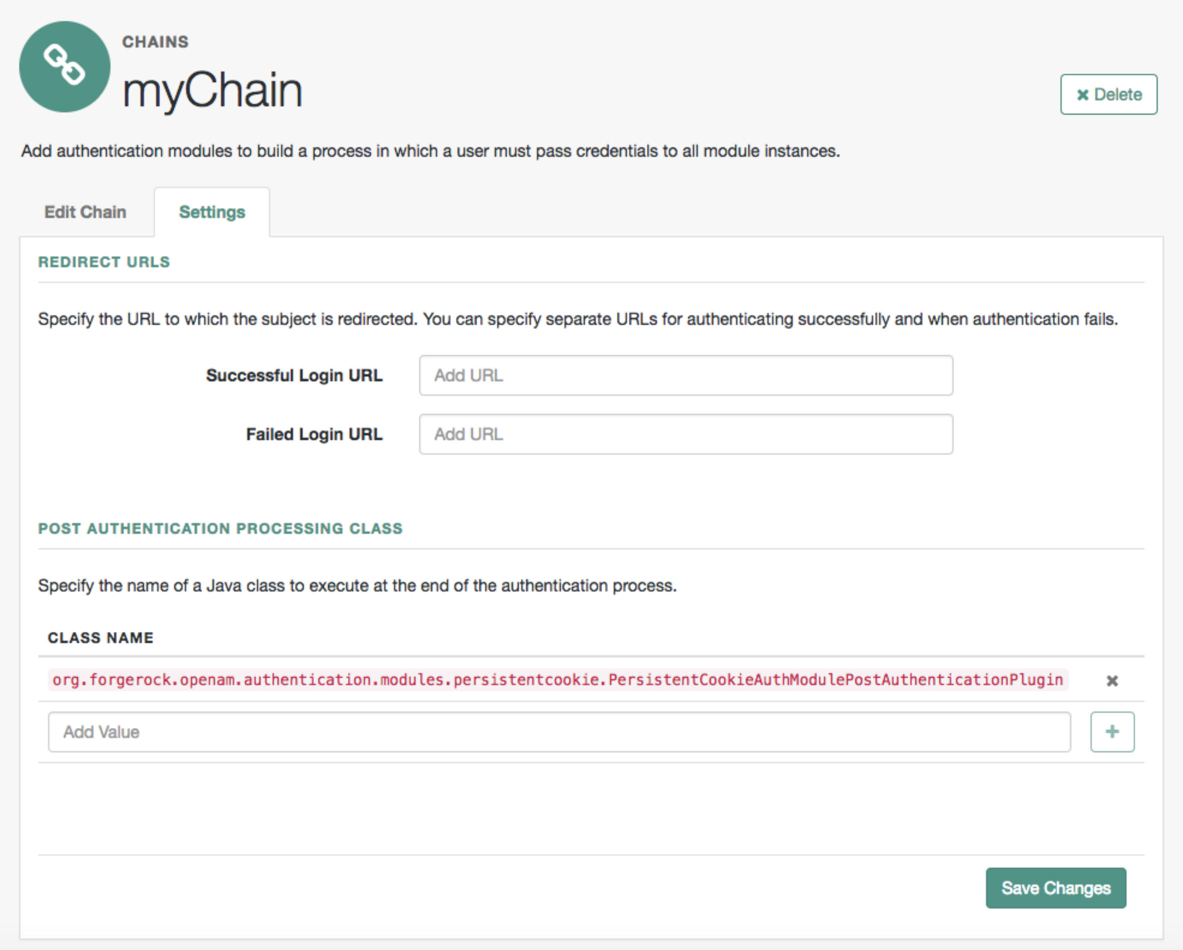
Successful authentication requires: At least one PASS flag No FAIL flags

1 PersistentCookieModule
Persistent Cookie
Sufficient

2 LDAP
LDAP
Required

Successful authentication requires: At least one PASS flag No FAIL flags

Select the Settings tab and locate settings for the post-authentication processing class. Set the Class Name to `org.forgerock.openam.authentication.modules.persistentcookie.PersistentCookieAuthModulePostAuthenticationPlugin`, as shown in the following figure:



CHAINS

myChain

Add authentication modules to build a process in which a user must pass credentials to all module instances.

REDIRECT URLS

Specify the URL to which the subject is redirected. You can specify separate URLs for authenticating successfully and when authentication fails.

Successful Login URL: Add URL

Failed Login URL: Add URL

POST AUTHENTICATION PROCESSING CLASS

Specify the name of a Java class to execute at the end of the authentication process.

CLASS NAME

`org.forgerock.openam.authentication.modules.persistentcookie.PersistentCookieAuthModulePostAuthenticationPlugin` ✖

Add Value +

You should now be able to authenticate automatically, as long as the cookie exists for the associated domain.

Tip

To configure the Persistent Cookie module globally in the AM console, navigate to Configure > Authentication, and then click Persistent Cookie.

For detailed information about this module's configuration properties, see "Persistent Cookie Authentication Module Properties".

RADIUS Authentication Module

The Remote Authentication Dial-In User Service (RADIUS) module lets AM authenticate users against RADIUS servers.

For detailed information about this module's configuration properties, see "RADIUS Authentication Module Properties".

SAE Authentication Module

The Secure Attribute Exchange (SAE) module lets AM authenticate a user who has already authenticated with an entity that can vouch for the user to AM, so that AM creates a session for the user. This module is useful in virtual federation, where an existing entity instructs the local AM instance to use federation protocols to transfer authentication and attribute information to a partner application.

For detailed information about this module's configuration properties, see "SAE Authentication Module Properties".

SAML2 Authentication Module

The SAML2 authentication module lets administrators integrate SAML v2.0 single sign-on and single logout into an AM authentication chain.

You use the SAML2 authentication module when deploying SAML v2.0 single sign-on in integrated mode. In addition to configuring SAML2 authentication module properties, integrated mode deployment requires that you make several changes to service provider configurations. Before attempting to configure a SAML2 authentication module instance, review "Implementing SAML v2.0 Single Sign-On in Integrated Mode" in the *SAML v2.0 Guide* and make sure that you have made any required changes to your service provider configuration.

For detailed information about this module's configuration properties, see "SAML2 Authentication Module Properties".

Scripted Authentication Module

A scripted authentication module runs scripts to authenticate a user. The configuration for the module can hold two scripts, one to include in the web page run on the client user-agent, another to run in AM on the server side.

The client-side script is intended to retrieve data from the user-agent. This must be in a language the user-agent can run, such as JavaScript, even if the server-side script is written in Groovy.

The server-side script is intended to handle authentication.

Scripts are stored not as files, but instead as AM configuration data. This makes it easy to update a script on one AM server, and then to allow replication to copy it to other servers. You can manage the scripts through the AM console, where you can write them in the text boxes provided or upload them from files.

You can also upload scripts and associate them with a scripted authentication module by using the **ssoadm** command.

The following example shows how to upload a server-side script from a file, create a scripted authentication module, and then associate the uploaded script with the new module.

```
#
# Upload a server-side script from a script file, myscript.groovy.
#

ssoadm create-sub-cfg \
--realm / \
--adminid amadmin \
--password-file /tmp/pwd.txt \
--servicename ScriptingService \
--subconfigname scriptConfigurations/scriptConfiguration \
--subconfigid myScriptId \
--attributevalues \
"name=My Scripted Auth Module Script" \
"script-file=myscript.groovy" \
"context=AUTHENTICATION_SERVER_SIDE" \
"language=GROOVY"
#
# Create a scripted authentication module, myScriptedAuthModule.
#

ssoadm create-auth-instance \
--realm / \
--adminid amadmin \
--password-file /tmp/pwd.txt \
--authtype Scripted \
--name myScriptedAuthModule

#
# Associate the script with the auth module, and disable client-side scripts.
#

ssoadm update-auth-instance \
--realm / \
--adminid amadmin \
--password-file /tmp/pwd.txt \
--name myScriptedAuthModule \
--attributevalues \
"iplanet-am-auth-scripted-server-script=myScriptId" \
"iplanet-am-auth-scripted-client-script-enabled=false"
```

If you have multiple separate sets of client-side and server-side scripts, then configure multiple modules, one for each set of scripts.

For details on writing authentication module scripts, see "Using Server-side Authentication Scripts in Authentication Modules".

For detailed information about this module's configuration properties, see "Scripted Authentication Module Properties".

SecurID Authentication Module

The SecurID module lets AM authenticate users with RSA Authentication Manager software and RSA SecurID authenticators.

> **Important**
>
> To use the SecurID authentication module, you must first build an AM `.war` file that includes the supporting library. For more information, see "Enabling RSA SecurID Support" in the *Installation Guide*.

For detailed information about this module's configuration properties, see "SecurID Authentication Module Properties".

Social Authentication Modules

The social authentication modules let AM authenticate clients of OAuth 2.0 or OpenID Connect 1.0 resource servers. References in this section are to RFC 6749, The OAuth 2.0 Authorization Framework.

AM provides pre-configured authentication modules for the following social identity providers:

- Instagram
- VKontakte
- WeChat

AM provides two authentication modules for the WeChat social identity provider. The *Social Auth WeChat* authentication module implements a login flow that requires the user to scan an on-screen QR code with the WeChat app. The *Social Auth WeChat Mobile* authentication module implements an alternative login flow for users authenticating on their mobile device, who would not be able to scan a QR code displayed on the mobile device's screen.

AM provides two generic authentication modules, one for OAuth 2.0, and another for OpenID Connect 1.0, for authenticating users of standards-compliant social identity providers, for example Facebook and Google.

> **Tip**
>
> A wizard for configuring common social authentication providers, such as Facebook, Google, and VKontakte, is available by navigating to Realms > *Realm Name* > Dashboard > Configure Social Authentication. For more information, see "Configuring Pre-Populated Social Authentication Providers".

If the social authentication module is configured to create an account when none exists, then you must provide valid SMTP settings in the Email tab. The social identity provider must also provide the user's email address. As part of account creation, the social authentication module sends the resource owner an email with an account activation code. To send email, AM uses the SMTP settings from the Email tab of the configuration of the social authentication module.

For detailed information about the social authentication module's configuration properties, see the following sections:

- "Social Authentication Module Properties - OAuth 2.0"
- "Social Authentication Module Properties - OpenID Connect 1.0"
- "Social Authentication Module Properties - Instagram"
- "Social Authentication Module Properties - VKontakte"
- "Social Authentication Module Properties - WeChat"
- "Social Authentication Module Properties - WeChat Mobile"

Windows Desktop SSO Authentication Module

The Windows Desktop SSO module uses Kerberos authentication. The user presents a Kerberos token to AM through the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) protocol. The Windows Desktop SSO authentication module enables desktop single sign on such that a user who has already authenticated with a Kerberos Key Distribution Center can authenticate to AM without having to provide the login information again. Users might need to set up Integrated Windows Authentication in Internet Explorer or Microsoft Edge to benefit from single sign on when logged on to a Windows desktop.

For detailed information about this module's configuration properties, see "Windows Desktop SSO Authentication Module Properties".

Warning

If you are using the Windows Desktop SSO module as part of an authentication chain and Windows Desktop SSO fails, you may no longer be able to `POST` data to non-NTLM-authenticated web sites. For information on a possible workaround, see *Microsoft knowledge base article KB251404*.

Configuring Authentication Chains

Once you have configured authentication modules and added the modules to the list of module instances, you can configure authentication chains. Authentication chains let you handle cases where alternate modules or credentials are needed. If you need modules in the chain to share user credentials, then set options for the module.

> **Tip**
>
> AM provides a wizard for configuring authentication providers, including Facebook, Google, and Microsoft. The wizard creates a relevant authentication chain as part of the process. For more information, see "*Implementing Social Authentication*".

To Create an Authentication Chain

1. On the Realms page of the AM console, click the realm for which to create the authentication chain.

2. On the Realm Overview page, click Authentication in the left-hand menu, and then click Chains.

3. On the Authentication Chains page, click Add Chain. Enter new chain name, and then click Create.

4. On the New Module dialog, select the authentication module in the chain, and then assign appropriate criteria (Optional, Required, Requisite, Sufficient) as described in "About Authentication Modules and Chains". You can also configure where AM redirects the user upon successful and failed authentication, and plug in your post-authentication processing classes as necessary.

5. (Optional) If you need modules in the chain to share user credentials, consider the following available options. Enter the key and its value, and then click Plus (+). When you finish entering the options, click OK.

   `iplanet-am-auth-store-shared-state-enabled`

   Set `iplanet-am-auth-store-shared-state-enabled=true` to store the credentials captured by this module in shared state. This enables subsequent modules in the chain to access the credentials captured by this module. The shared state is cleared when the user successfully authenticates, quits the chain, or logs out.

   Default: `true`

   > **Note**
   >
   > OATH and OTP codes are never added to the shared state, and cannot be shared between other modules in the chain.

   `iplanet-am-auth-shared-state-enabled`

   Set `iplanet-am-auth-shared-state-enabled=true` to allow this module to access the credentials, such as user name and password, that have been stored in shared state by previous modules in the authentication chain.

   Default: `false`

iplanet-am-auth-shared-state-behavior-pattern

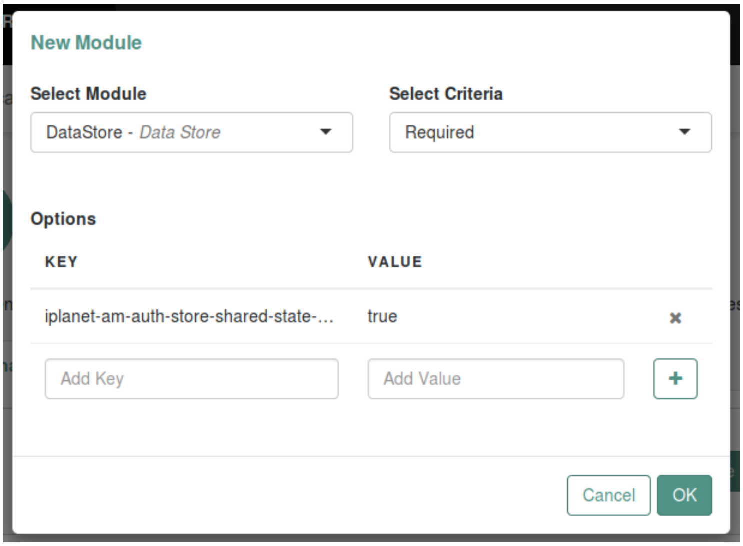
Set `iplanet-am-auth-shared-state-behavior-pattern=tryFirstPass` to try authenticating with the username and password stored in shared state. If authentication fails, AM displays the login screen of this module for the user to re-enter their credentials.

Set `iplanet-am-auth-shared-state-behavior-pattern=useFirstPass` to prevent the user from entering the username and password twice during authentication. Typically, you set the property to `useFirstPass` for all modules in the chain except the first module. If authentication fails, then the module fails.

Default: `tryFirstPass`

For example, consider a chain with two modules sharing credentials according to the following settings: the first module in the chain has the option `iplanet-am-auth-store-shared-state-enabled=true`, and criteria `REQUIRED`.

Authentication Chain First Module



New Module

Select Module: DataStore - *Data Store*

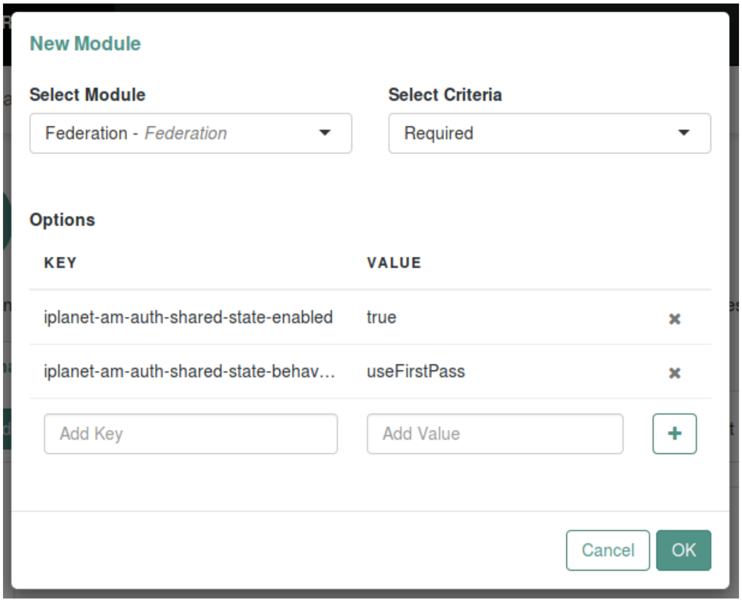
Select Criteria: Required

Options

KEY	VALUE
iplanet-am-auth-store-shared-state-...	true

The second module in the chain has options `iplanet-am-auth-shared-state-enabled=true`, `iplanet-am-auth-shared-state-behavior-pattern=useFirstPass` with criteria `REQUIRED`.

Authentication Chain Second Module



New Module

Select Module: Federation - *Federation*

Select Criteria: Required

Options

KEY	VALUE
iplanet-am-auth-shared-state-enabled	true
iplanet-am-auth-shared-state-behav…	useFirstPass

6. Click Save Changes.

   The following authentication sequence would occur: the user enters their credentials for the first module and successfully authenticates. The first module shares the credentials with the second module, successfully authenticating the user without prompting again for their credentials, unless the credentials for the first module do not successfully authenticate the user to the second module.

Implementing Post-Authentication Plugins

Post-authentication plugins (PAP) let you include custom processing at the end of the authentication process and when users log out of AM.

In the AM console, you add post-authentication plugins to an authentication chain. Navigate to Realms > *Realm Name* > Authentication > Chains > *Auth Chain Name* > Settings > Post Authentication Processing Class > Class Name.

See "Creating Post-Authentication Plugins for Chains" for more information about post authentication plugins.

Standard Post-Authentication Plugins

AM provides some post-authentication plugins as part of the standard product delivery.

Class name: `org.forgerock.openam.authentication.modules.adaptive.AdaptivePostAuthenticationPlugin`

> The adaptive authentication plugin serves to save cookies and profile attributes after successful authentication.
>
> Add it to your authentication chains that use the adaptive authentication module configured to save cookies and profile attributes.

Class name: `org.forgerock.openam.authentication.modules.common.JaspiAuthLoginModulePostAuthenticationPlugin`

> The Java Authentication Service Provider Interface (JASPI) post authentication plugin initializes the underlying JASPI `ServerAuth` module.
>
> JASPI defines a standard service provider interface (SPI) where developers can write message level authentication agents for Java containers on either the client side or the server side.

Class name: `org.forgerock.openam.authentication.modules.oauth2.OAuth2PostAuthnPlugin`

> The OAuth 2.0 post-authentication plugin builds a global logout URL used by `/oauth2c/OAuthLogout.jsp` after successful OAuth 2.0 client authentication. This logs the resource owner out with the OAuth 2.0 provider when logging out of AM.
>
> Before using this plugin, configure the OAuth 2.0 authentication module with the correct OAuth 2.0 Provider logout service URL, and set the Logout options to Log out or Prompt. This plugin cannot succeed unless those parameters are correctly set.
>
> Sometimes OAuth 2.0 providers change their endpoints, including their logout URLs. When using a provider like Facebook, Google, or MSN, make sure you are aware when they change their endpoint locations so that you can change your client configuration accordingly.

Class name: `org.forgerock.openam.authentication.modules.saml2.SAML2PostAuthenticationPlugin`

> The SAML v2.0 post-authentication plugin that gets activated for single logout. Supports HTTP-Redirect for logout-sending messages only.
>
> Set the post-authentication processing class for the authentication chain that contains the SAML v2.0 authentication module.

Class name: `org.forgerock.openam.authentication.modules.persistentcookie.PersistentCookieAuthModule`

> The Persistent Cookie Authentication Module provides logic for persistent cookie authentication in AM. It makes use of the JASPI `JwtSession` module to create and verify the persistent cookie.

Class name: `com.sun.identity.authentication.spi.ReplayPasswd`[2]

Password replay post-authentication plugin class that uses a DES/ECB/NoPadding encryption algorithm. This class is deprecated in favor of the `com.sun.identity.authentication.spi.JwtReplayPassword` class.

The plugin encrypts the password captured by AM during the authentication process and stores it in a session property. IG or a web agent looks up for the property, decrypts it, and replays the password into legacy applications.

To configure password replay for AM and IG, see the *ForgeRock Identity Gateway Gateway Guide*.

Class name: `com.sun.identity.authentication.spi.JwtReplayPassword`[2]

Password replay post-authentication plugin class that uses a JWT-based AES A128CBC-HS256 encryption algorithm.

The plugin encrypts the password captured by AM during the authentication process and stores it in a session property. IG looks up for the property, decrypts it, and replays the password into legacy applications.

Only IG 6 or later is supported.

To configure password replay for AM and IG, see the *ForgeRock Identity Gateway Gateway Guide*.

If necessary, you can also write your own custom post-authentication plugin as described in "Creating Post-Authentication Plugins for Chains".

Configuring the Default Authentication Tree or Chain

By default, AM configures a default authentication chain, `ldapService`, which uses the `DataStore` module for authentication. This default authentication chain is configured for both administrators and non-administrators out-of-the-box after installation.

> **Warning**
>
> Special care must be given when setting your *default* authentication tree or chain.
>
> If you leave the default authentication to the `ldapService` chain, the user can still post their username and password into the authentication endpoint to retrieve a session, regardless of the services configured for authentication.
>
> For example, consider a deployment where you disable module-based authentication and keep the default authentication chain to the out-of-the-box `ldapStore` authentication chain using `DataStore` module. If you have

[2] Only one password replay post-authentication plugin class can be active for a given AM deployment.

set up two factor authentication for your users, your users can still access their accounts without performing the correct two factor authentication chain login sequence by using the default `ldapService` chain.

When you set the default authentication tree or chain, make sure it is set to your most secure tree or chain once you are ready to go to production and not left to the default `ldapService` chain.

To Set the Default Authentication Tree or Chain

1. Before you select the default tree or chain for users, and especially for administrators, test the authentication tree or chain first.

   When making a request to the XUI, specify the realm or realm alias as the value of a `realm` parameter in the query string, or the DNS alias in the domain component of the URL. If you do not use a realm alias, then you must specify the entire hierarchy of the realm, starting at the top-level realm. For example `https://openam.example.com:8443/openam/XUI/?realm=/customers/europe#login/`.

   For example, to test an authentication chain named `NewChain` in a subrealm called `subrealm`, the URL would be: `https://openam.example.com:8443/openam/XUI/?realm=/subrealm&service=NewChain#login`. If you cannot log in, then go back and fix the authentication chain's configuration before making it the default.

2. Navigate to Realms > *Realm Name* > Authentication > Settings > Core.

3. Adjust the Administrator Authentication Configuration drop-down to the required tree or chain. Administrative users, such as `amAdmin`, use this tree or chain to log in.

   By default, `amAdmin` can log in at `/openam/XUI/#Login`. You can change the URL for your deployment.

4. Adjust the Organization Authentication Configuration drop-down to the required tree or chain. Non-administrative users use this tree or chain to log in.

5. Save your work.

Chapter 3
Implementing Social Authentication

Social authentication refers to AM's ability to delegate authentication through third-party identity providers, such as Facebook and Google, and other third-party providers.

AM allows delegation of authentication by providing provider-specific, and also generic OAuth 2.0 and OpenID Connect 1.0 authentication modules. For background information, see "About Social Authentication".

Note

To allow AM to contact internet services through a proxy, see "Settings for Configuring a JVM Proxy" in the *Installation Guide*.

Moreover, you can control the behavior of the connection factory that AM uses as a client of the social identity providers:

+ *Client Connection Handler Properties*

The following advanced server properties control different aspects of the connection factory:

- `org.forgerock.openam.httpclienthandler.system.clients.connection.timeout`
- `org.forgerock.openam.httpclienthandler.system.clients.max.connections`
- `org.forgerock.openam.httpclienthandler.system.clients.pool.ttl`
- `org.forgerock.openam.httpclienthandler.system.clients.response.timeout`
- `org.forgerock.openam.httpclienthandler.system.clients.retry.failed.requests.enabled`
- `org.forgerock.openam.httpclienthandler.system.clients.reuse.connections.enabled`

They have sensible defaults configured, but if you need to change them, see see "Advanced Properties" in the *Reference*.

This chapter explains the server configuration required to implement social authentication in AM:

- "Configuring Pre-Populated Social Authentication Providers"
- "Configuring Custom Social Authentication Providers"
- "Configuring the Social Authentication Implementations Service"

Configuring Pre-Populated Social Authentication Providers

AM provides wizards to quickly enable authentication with Facebook, Google, and VKontakte. Most settings are pre-populated, only a *Client ID* and *Client Secret* are required.

To obtain a *Client ID* and *Client Secret* you should register an application with the third party provider, at the following links:

Facebook

Facebook App Quickstart

Google

Google Developers Console

> **Note**
>
> You must enable the Google+ API in order to authenticate with Google. To enable the Google+ API, login to the Google Developers Console, select your project, navigate to APIs and auth > APIs, and then set the status of the `Google+ API` to `ON`.

VKontakte

VKontakte Developers - My apps

To Configure Pre-Populated Social Authentication Providers

Once you have registered an application and obtained credentials from the social authentication provider, follow the steps below to configure authentication with the provider:

1. Select Realms > *Realm Name* > Dashboard > Configure Social Authentication, and then click the link for the social authentication provider you want to configure—*Configure Facebook Authentication*, *Configure Google Authentication*, or *Configure VKontakte Authentication*.

2. On the configure third party authentication page:

   a. Select the realm in which to enable social authentication.

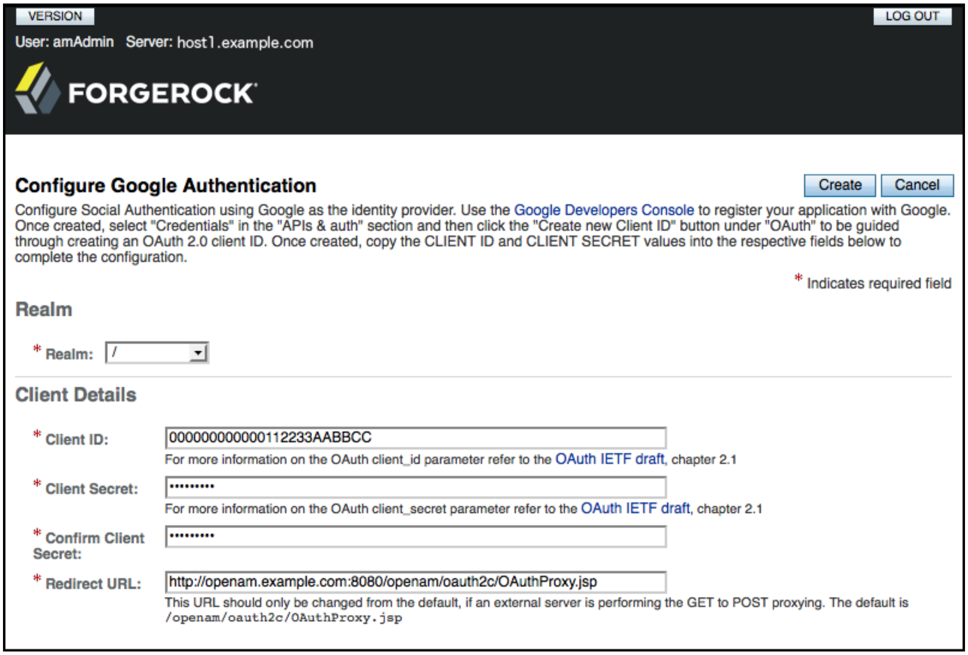
   b. Enter the *Client ID* obtained from the third party authentication provider.

   c. Enter the *Client Secret* obtained from the third party authentication provider, and repeat it in the `Confirm Client Secret` field.

   d. Leave the default `Redirect URL`, unless you are using an external server as a proxy.

   e. Click `Create`.

The Configure Google Authentication Wizard



User: amAdmin Server: host1.example.com

Configure Google Authentication

Configure Social Authentication using Google as the identity provider. Use the Google Developers Console to register your application with Google. Once created, select "Credentials" in the "APIs & auth" section and then click the "Create new Client ID" button under "OAuth" to be guided through creating an OAuth 2.0 client ID. Once created, copy the CLIENT ID and CLIENT SECRET values into the respective fields below to complete the configuration.

* Indicates required field

Realm

* Realm: /

Client Details

* Client ID: 000000000000112233AABBCC
For more information on the OAuth client_id parameter refer to the OAuth IETF draft, chapter 2.1

* Client Secret: •••••••••
For more information on the OAuth client_secret parameter refer to the OAuth IETF draft, chapter 2.1

* Confirm Client Secret: •••••••••

* Redirect URL: http://openam.example.com:8080/openam/oauth2c/OAuthProxy.jsp
This URL should only be changed from the default, if an external server is performing the GET to POST proxying. The default is `/openam/oauth2c/OAuthProxy.jsp`

On completion, the wizard displays a message confirming the successful creation of a new authentication module and an authentication chain for the provider, and either the creation of a new Social Authentication Implementations service named `socialAuthNService`, or an update if it already existed.

You can configure the authentication module, authentication chain, and Social Authentication Implementations service that you created by using the wizards in the same way as manually created versions. For more information, see "Configuring Authentication Modules", "Configuring Authentication Chains", and "Configuring the Social Authentication Implementations Service".

Integrating Social Authentication with Identity Management

The wizards configure the settings for logging in to AM using social identity providers such as Google, Facebook, and VKontakte.

To use AM social authentication as part of an IDM deployment, some additional configuration of the created authentication modules is required.

To Integrate Social Authentication with Identity Management

After using the social authentication wizard, perform the following additional steps to configure AM to work with an IDM deployment:

1. When using **Google** as the social identity provider, in the AM console navigate to Realms > Realm Name > Authentication > Modules > GoogleSocialAuthentication.

   Modify the configuration as follows:

   a. Add `sub=iplanet-am-user-alias-list` to the Account Mapper Configuration property.

      The `iplanet-am-user-alias-list` property defines one or more aliases for mapping a user's multiple profiles.

   b. Add `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper|iplanet-am-user-alias-list|google-` to the Attribute Mapper property.

   c. Add `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper|iplanet-am-user-alias-list|google-` to the Attribute Mapper property.

2. When using **Facebook** as the social identity provider, in the AM console navigate to Realms > Realm Name > Authentication > Modules > FacebookSocialAuthentication.

   Modify the configuration as follows:

   a. Add `id=iplanet-am-user-alias-list` to the Account Mapper Configuration property.

      The `iplanet-am-user-alias-list` property defines one or more aliases for mapping a user's multiple profiles.

   b. Add `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper|iplanet-am-user-alias-list|facebook-` to the Attribute Mapper property.

3. Enable the Create account if it does not exist property.

4. Save your changes.

Configuring Custom Social Authentication Providers

AM provides a wizard to quickly enable authentication with any third party provider that supports the *OpenID Connect Discovery 1.0 specification*.

Tip

To configure social authentication providers that use OAuth 2.0, such as Microsoft, configure a Social Auth OAuth2 authentication module.

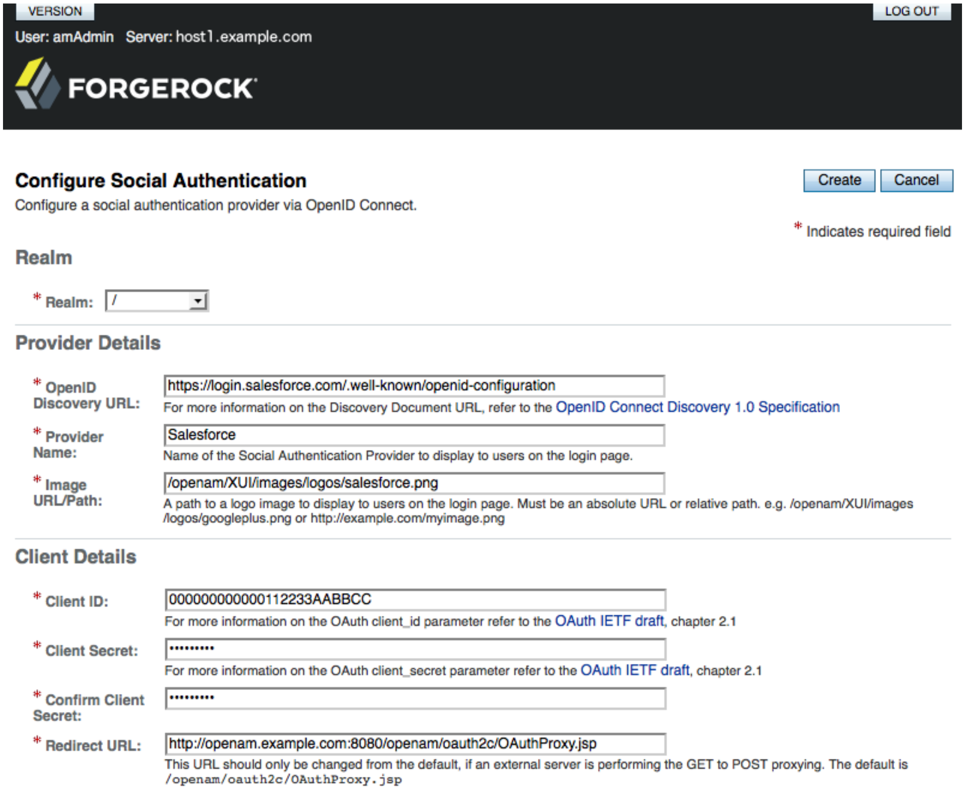
To Configure Custom Social Authentication Providers

You must first register an application with the third party provider to obtain a *Client ID*, *Client Secret*, and the *OpenID Discovery URL*.

Once you have registered an application and obtained your credentials from the social authentication provider, follow the steps below to configure authentication with the provider:

1. Select Realms > *Realm Name* > Dashboard > Configure Social Authentication, and then click the *Configure Other Authentication* link.
2. On the configure social authentication page:
   a. Select the realm in which to enable social authentication.
   b. Enter the *OpenID Discovery URL* obtained from the third party authentication provider.
   c. Enter a name for the provider in the `Provider Name` field. AM uses this as a label on the login page to identify the provider.
   d. Enter the URL of an image to be used on the login page in the `Image URL` field. AM places the image on the login page, to enable authentication with the provider.
   e. Enter the *Client ID* obtained from the third party authentication provider.
   f. Enter the *Client Secret* obtained from the third party authentication provider, and repeat it in the `Confirm Client Secret` field.
   g. Leave the default `Redirect URL`, unless you are using an external server as a proxy.
   h. Click `Create`.

The Configure Social Authentication Wizard



User: amAdmin Server: host1.example.com

Configure Social Authentication

Configure a social authentication provider via OpenID Connect.

* Indicates required field

Realm

* Realm: /

Provider Details

* OpenID Discovery URL: https://login.salesforce.com/.well-known/openid-configuration
For more information on the Discovery Document URL, refer to the OpenID Connect Discovery 1.0 Specification

* Provider Name: Salesforce
Name of the Social Authentication Provider to display to users on the login page.

* Image URL/Path: /openam/XUI/images/logos/salesforce.png
A path to a logo image to display to users on the login page. Must be an absolute URL or relative path. e.g. /openam/XUI/images/logos/googleplus.png or http://example.com/myimage.png

Client Details

* Client ID: 000000000000112233AABBCC
For more information on the OAuth client_id parameter refer to the OAuth IETF draft, chapter 2.1

* Client Secret: •••••••••
For more information on the OAuth client_secret parameter refer to the OAuth IETF draft, chapter 2.1

* Confirm Client Secret: •••••••••

* Redirect URL: http://openam.example.com:8080/openam/oauth2c/OAuthProxy.jsp
This URL should only be changed from the default, if an external server is performing the GET to POST proxying. The default is `/openam/oauth2c/OAuthProxy.jsp`

On completion, the wizard displays a message confirming the successful creation of a new authentication module and an authentication chain for the provider, and either the creation of a new Social Authentication Implementations service named `socialAuthNService`, or an update if it already existed.

You can configure the authentication module, authentication chain, and Social Authentication Implementations service that you created by using the wizard in the same way as manually created versions. For more information, see "Configuring Authentication Modules", "Configuring Authentication Chains", and "Configuring the Social Authentication Implementations Service".

Configuring the Social Authentication Implementations Service

You can add logos to the login page to allow users to authenticate using configured social authentication providers.

Wizards are provided to configure common social authentication providers, which also configure the Social Authentication Implementations Service to add logos to the login page. You can manually add other authentication chains that contain an social authentication modules.

To add a social authentication provider to the login screen, you must first configure a social authentication module, and an authentication chain that contains it:

- Use a wizard. See "Configuring Pre-Populated Social Authentication Providers" and "Configuring Custom Social Authentication Providers".
- Configure the Social Authentication Implementations Service, and then create an authentication module and a chain. See "To Configure the Social Authentication Implementations Service", "Configuring Authentication Modules" and "Configuring Authentication Chains".

To Configure the Social Authentication Implementations Service

Once you have created an authentication chain containing a social authentication module, perform the following steps to add a logo for the authentication provider to the AM login screen:

1. On the Realms page of the AM console, click the realm containing the authentication module and authentication chain to be added to the login screen.
2. On the Services page for the realm:
   - If the `Social Authentication Implementations Service` exists, click on it.
   - If the `Social Authentication Implementations Service` does not exist, click Add a Service, and then select Social Authentication Implementations, and then click Create.
3. On the Social Authentication Implementations page:

   a. In the *Display Names* section, enter a Map Key, enter the text to display as ALT text on the logo in the Corresponding Map Value field, and then click Add.

   > **Note**
   >
   > AM uses the value in the Map Key fields throughout the configuration to tie the various implementation settings to each other. The value is case-sensitive.

   b. In the *Authentication Chains* section, re-enter the Map Key used in the previous step, select the authentication chain from the Corresponding Map Value list, and then click Add.

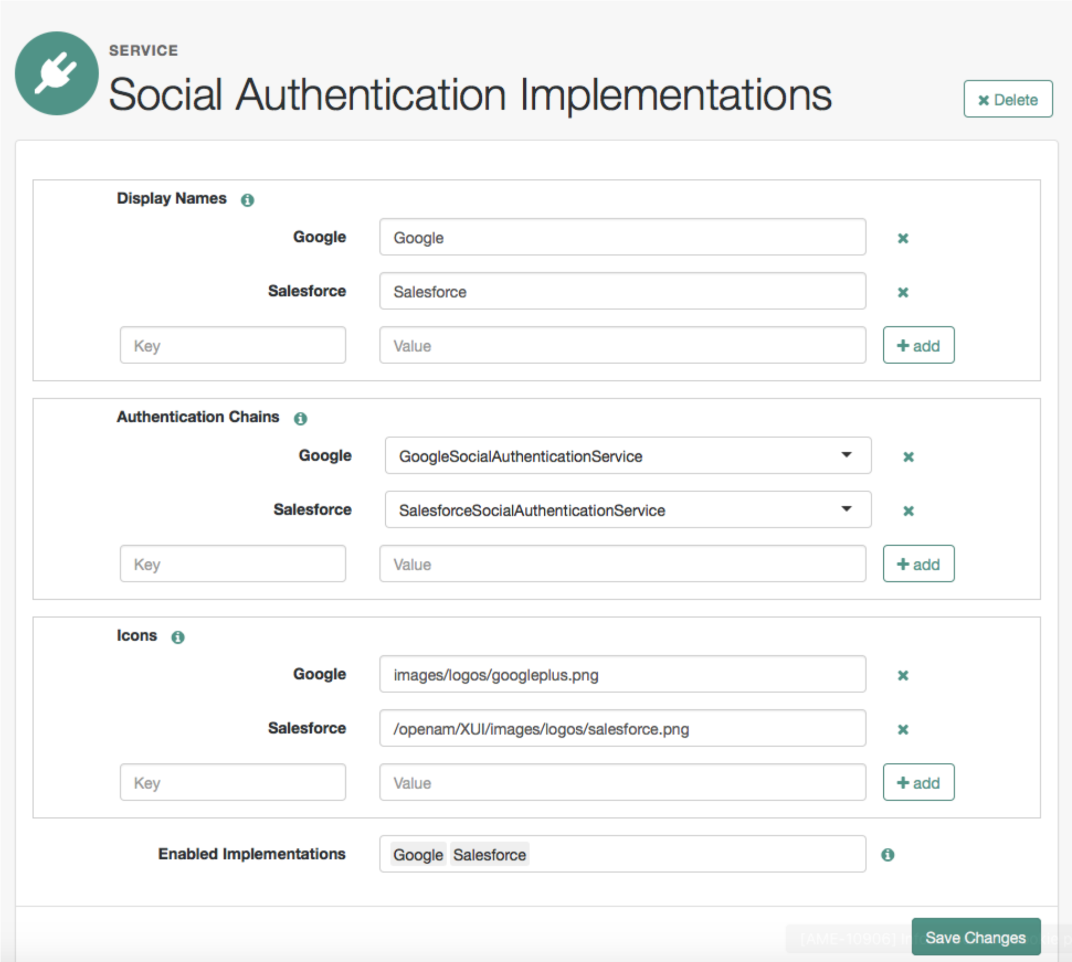
c. In the *Icons* section, re-enter the Map Key used in the previous steps, enter the path to a logo image to be used on the login screen in the Corresponding Map Value list, and then click Add.

d. In the *Enabled Implementations* field, re-enter the Map Key used in the previous steps, and then click Add.

> **Tip**
>
> Removing a Map Key from the Enabled Implementations list removes the associated logo from the login screen. There is no need to delete the Display Name, Authentication Chain or Icon configuration to remove the logo from the login screen.

e. Click Save Changes.

Configuring the Social Authentication Implementations service



SERVICE

Social Authentication Implementations

Display Names
- Google: Google
- Salesforce: Salesforce

Authentication Chains
- Google: GoogleSocialAuthenticationService
- Salesforce: SalesforceSocialAuthenticationService

Icons
- Google: images/logos/googleplus.png
- Salesforce: /openam/XUI/images/logos/salesforce.png

Enabled Implementations: Google Salesforce

An icon now appears on the AM login screen, allowing users to authenticate with the third party authentication provider.

Chapter 4

Implementing Multi-Factor Authentication

Multi-factor authentication is a security process that requires users to provide more than one form of credentials when logging in or accessing a resource. A common multi-factor authentication scenario is for users to submit a user ID and password, and then submit a one-time password generated by an authenticator application on their mobile phone to access a resource.

This chapter covers how administrators implement and support multi-factor authentication, and how end users authenticate using multi-factor authentication. See the following sections:

- "Configuring Multi-Factor Authentication Service Settings"
- "Letting Users Opt Out of One-Time Password Authentication"
- "Creating Multi-Factor Authentication Trees"
- "Creating Multi-Factor Authentication Chains"
- "Managing Devices for Multi-Factor Authentication"
- "Authenticating Using Multi-Factor Authentication"

For conceptual information about multi-factor authentication, see "About Multi-Factor Authentication".

Configuring Multi-Factor Authentication Service Settings

AM provides a number of services that must be configured to provide multi-factor authentication with the ForgeRock Authenticator app.

The service for customizing one-time password implementation is:

ForgeRock Authenticator (OATH) Service

> Specifies the attribute in which to store information about the registered OATH device, and whether to encrypt that information.
>
> Also specifies the attribute used to indicate if a user has opted out of one-time passwords.
>
> For detailed information about the available properties, see "ForgeRock Authenticator (OATH) Service" in the *Reference*.

WebAuthn Profile Encryption Service

> Specifies the attribute in which to store information about registered WebAuthn devices, and whether to encrypt that information.

For detailed information about the available properties, see "WebAuthn Profile Encryption Service" in the *Reference*.

The services required for implementing push notifications are:

ForgeRock Authenticator (Push) Service

Specifies the attribute in which to store information about the registered Push device, and whether to encrypt the data.

For detailed information about the available properties, see "ForgeRock Authenticator (Push) Service" in the *Reference*.

Push Notification Service

Configures how AM sends push notifications to registered devices, including endpoints, and access credentials.

For information on provisioning the credentials required by the Push Notification Service, see How To Configure Service Credentials (Push Auth, Docker) in Backstage in the *ForgeRock Knowledge Base*.

For detailed information about the available properties, see "Push Notification Service" in the *Reference*.

To configure these services globally for an AM deployment, navigate to Configure > Global Services, and then click the service to configure.

To configure these services for a realm, navigate to Realms > *Realm Name*, and then click Services. Add an instance of the service to the realm and configure settings in the service as required.

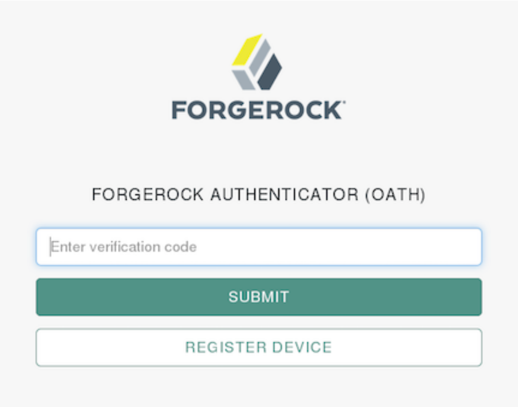
Letting Users Opt Out of One-Time Password Authentication

Letting users opt out of providing one-time passwords when they perform multi-factor authentication is an important implementation decision. The Two Factor Authentication Mandatory setting under Realms > *Realm Name* > Authentication > Settings > General configures whether users can opt out.

When the Two Factor Authentication Mandatory setting is enabled, users must provide a one-time password every time they authenticate to a chain that includes a ForgeRock Authenticator (OATH) authentication module. When the setting is disabled, the user can optionally skip one-time passwords.

By default, AM lets users opt out of providing one-time passwords. Users authenticating with one-time passwords for the first time are prompted with a screen that lets them opt out of providing one-time passwords.

With the Two Factor Authentication Mandatory setting enabled, the user experience differs from the default behavior. AM does not provide an option to skip multi-factor authentication during the initial attempt at multi-factor authentication:


FORGEROCK AUTHENTICATOR (OATH)

When configuring an authentication chain that implements one-time passwords, you need to be aware that a user's decision to opt out affects the authentication process. When a user who has opted out of providing one-time passwords authenticates to a chain that includes a ForgeRock Authenticator (OATH) authentication module, that module *always* passes authentication.

Consider the example authentication chain in "Creating Authentication Chains for One-Time Password Authentication". The first authentication module is a Data Store module and the second authentication module is a ForgeRock Authenticator (OATH) module. Both authentication modules have the Requisite flag setting.

A user who has opted out of providing one-time passwords might experience the following sequence of events when authenticating to the chain:

1. The Data Store authentication module prompts the user to provide a user ID and password.
2. The user provides a valid user ID and password.
3. Data Store authentication passes, and authentication proceeds to the next module in the chain—the ForgeRock Authenticator (OATH) module.
4. The ForgeRock Authenticator (OATH) authentication module determines that the user has opted out of providing one-time passwords.
5. ForgeRock Authenticator (OATH) authentication passes. Because it is the last authentication module in the chain, AM considers authentication to have completed successfully.

Contrast the preceding sequence of events to the experience of a user who has not opted out of providing one-time passwords, or who is required to provide one-time passwords, while authenticating to the same chain:

1. The Data Store authentication module prompts the user to provide a user ID and password.
2. The user provides a valid user ID and password.

3. Data Store authentication passes, and authentication proceeds to the next module in the chain—the ForgeRock Authenticator (OATH) module.

4. The ForgeRock Authenticator (OATH) authentication module determines that the user has not opted out of providing one-time passwords, and prompts the user for a one-time password.

5. The user obtains a one-time password from the authenticator app on their mobile phone.

6. If the one-time password is valid, ForgeRock Authenticator (OATH) authentication passes. Because it is the last authentication module in the chain, AM considers authentication to have completed successfully. However, if the one-time password is not valid, ForgeRock Authenticator (OATH) authentication fails, and AM considers authentication to have failed.

Creating Multi-Factor Authentication Trees

The following sections provide steps for creating authentication trees that implement multi-factor authentication.

Creating Trees for Web Authentication (WebAuthn)

This section explains how to create an authentication tree to authenticate users by using a WebAuthn device, and allow them to register a device if they have not already done so.

If the user has already registered a WebAuthn device, they only need to enter their username, and then perform the authorization gesture with their registered device to access their profile.

If the user does not have a registered device, they are prompted for their password, and must be verified by the Data Store Decision Node before registering a new WebAuthn device. Once completed, they must authenticate with the new device before gaining access to their profile page.

To Create an Authentication Tree for WebAuthn Registration and Authentication

To create a multi-factor authentication tree for WebAuthn authentication, and registration if required, perform the following steps:

> **Note**
>
> The tree created in this procedure is an example, and does not provide user-friendly features, such as allowing retries of the users' password.

1. Log in to the AM console as an AM administrator, for example, `amadmin`.

2. Select the realm that will contain the authentication tree.

3. Create the authentication tree as follows:

a. Select Authentication > Trees, and then click Create Tree.

   The New Tree page appears.

b. Specify a name of your choosing, for example, `myWebAuthnTree`, and then click Create.

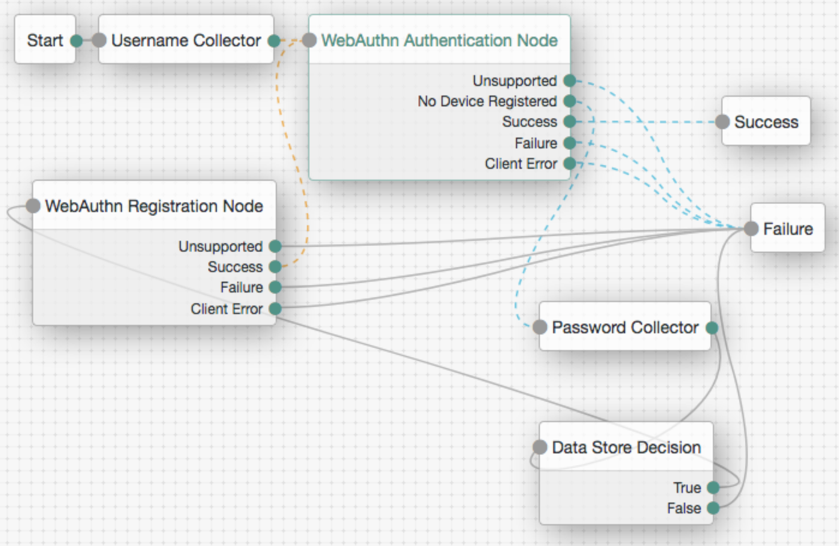
   The authentication tree designer is displayed, with the Start entry point connected to the Failure exit point.

   You can add nodes to the authentication tree by dragging the node from the Components panel on the left-hand side and dropping it into the designer area.

c. Add the following nodes to the authentication tree:

   - Username Collector Node
   - WebAuthn Authentication Node
   - Password Collector Node
   - Data Store Decision Node
   - WebAuthn Registration Node

d. Connect the nodes as demonstrated in the following figure:



e. Save your changes.

4. Test your WebAuthn authentication and registration tree as follows:

a. Logout of AM, and then navigate to a URL similar to the following: `https://openam.example.com:8443/openam/XUI/?realm=/&service=`*`myWebAuthnTree`*`#login`

> **Important**
>
> You must connect over HTTPS in order to use Web Authentication.

A login screen prompting you to enter your user ID appears.

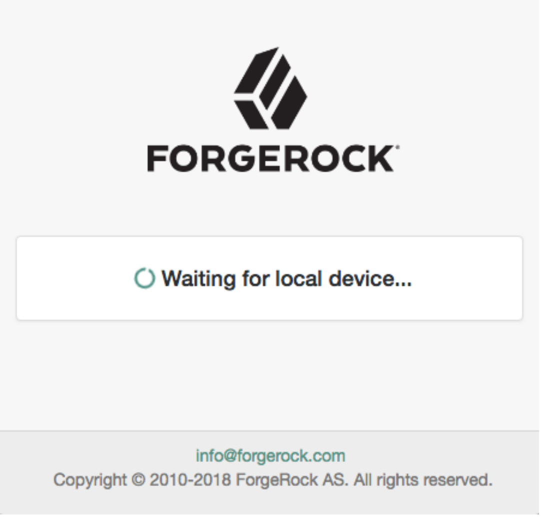
b. Enter the username of an existing account in the specified realm. For example, enter `demo`.

c. (Optional) If the `demo` user does not have a registered device:

   i. When asked for the user's password, enter the default `changeit`.

   ii. At the following screen, register a WebAuthn authenticator by performing an authorization gesture, for example press the button on a connected Yubikey.

The WebAuthn Registration node waiting for an authenticator.



Waiting for local device...

info@forgerock.com
Copyright © 2010-2018 ForgeRock AS. All rights reserved.

> **Note**
>
> The user's browser may present a consent pop-up to allow access to the authenticators available on the client. When consent has been granted, the browser activates the relevant authenticators, ready for registration.

If the device registration is successful, the user is redirected to the new node in the tree in order to authenticate with the newly registered device.

d. When prompted, authenticate to AM by performing an authorization gesture with a registered device.

   If the authorization is verified, the user's profile page is displayed.

   - Click the Dashboard link to see a list of the registered WebAuthn authenticators, and to rename or delete them. The default name for a new device is `New Security Key`.

Creating Authentication Trees for Push Authentication

Push authentication uses the authentication trees to receive push notifications and to perform the actual authentication itself.

Authentication trees can also be used for passwordless authentication using push notifications. When configured for passwordless authentication, the authentication flows asks the user to enter their user ID but not their password. A push notification is then sent to their registered device to complete the authentication by using the ForgeRock Authenticator app.

To Create an Authentication Tree for Push Authentication

The procedure assumes the following:

- Users will provide user IDs and passwords as the first step of multi-factor authentication.
- This procedure assumes users have registered using an authentication chain. At this time, authentication trees do not support registering devices. For more information, see in "To Create an Authentication Chain for Push Authentication".
- A push notification will be sent to the device as a second factor to complete authentication.

To create a multi-factor authentication tree, perform the following steps:

1. Log in to the AM console as an AM administrator, for example `amadmin`.
2. Select the realm that will contain the authentication tree.
3. Create the authentication tree as follows:

a. Select Authentication > Trees, and then click Create Tree.

The New Tree page appears.

b. Specify a name of your choosing, for example `myPushAuthTree`, and then click Create.

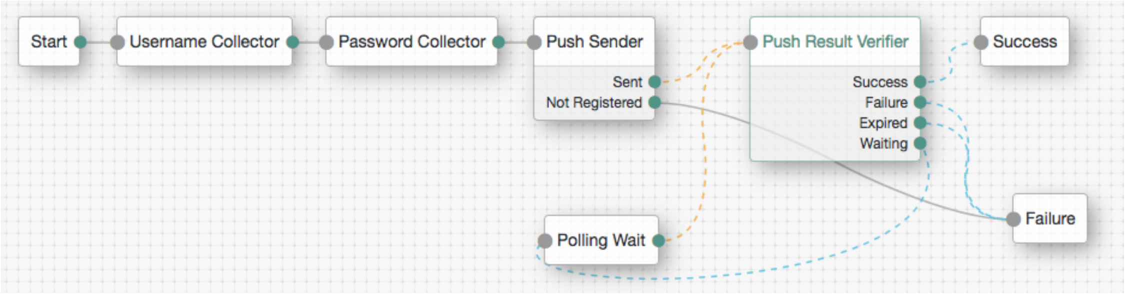
The authentication tree designer is displayed, with the Start entry point connected to the Failure exit point.

You can add nodes to the authentication tree by dragging the node from the Components panel on the left-hand side and dropping it into the designer area.

c. Add the following nodes to the authentication tree:

- Username Collector Node
- Password Collector Node
- Push Sender Node
- Push Result Verifier Node
- Polling Wait Node
- Success URL Node

d. Connect the nodes as demonstrated in the following figure:



e. Save your changes.

4. Test your authentication tree as follows:

a. Logout of AM, and then navigate to a URL similar to the following: `http://openam.example.com:8443/openam/XUI/?realm=/&service=`*`myPushAuthTree`*`#login`

A login screen prompting you to enter your user ID and password appears.

b. Follow the procedure described in "To Perform Authentication using a One-Time Password" to verify that you can use the ForgeRock Authenticator app to perform multi-factor authentication. If the authentication tree is correctly configured, authentication is successful and AM displays the user profile page.

To Create an Authentication Tree for Passwordless Authentication

The procedure assumes the following:

- Users will provide only their user IDs as the first step of multi-factor authentication.
- This procedure assumes users have registered using an authentication chain. At this time, authentication trees do not support registering devices. For more information, see in "To Create an Authentication Chain for Push Authentication".
- A push notification will be sent to the device as a second factor to complete authentication, without the need to enter the user's password.

To create a multi-factor authentication tree for passwordless authentication, perform the following steps:

1. Log in to the AM console as an AM administrator, for example `amadmin`.
2. Select the realm that will contain the authentication tree.
3. Create the authentication tree as follows:
    a. Select Authentication > Trees, and then click Create Tree.

       The New Tree page appears.

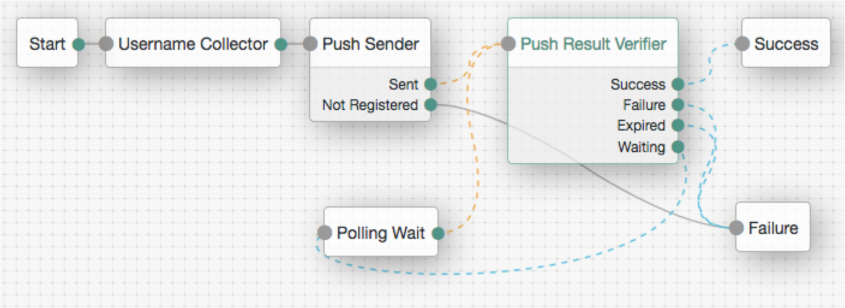
    b. Specify a name of your choosing, for example `myPasswordlessAuthTree`, and then click Create.

       The authentication tree designer is displayed, with the Start entry point connected to the Failure exit point.

       You can add nodes to the authentication tree by dragging the node from the Components panel on the left-hand side and dropping it into the designer area.

    c. Add the following nodes to the authentication tree:
        - Username Collector Node
        - Push Sender Node
        - Push Result Verifier Node
        - Polling Wait Node
        - Success URL Node

    d. Connect the nodes as demonstrated in the following figure:



e. Save your changes.

4. Test your authentication tree as follows:

   a. Logout of AM, and then navigate to a URL similar to the following: `http://openam.example.com:8443/openam/XUI/?realm=/&service=myPasswordlessAuthTree/#login`

      A login screen prompting you to enter your user ID appears.

   b. Follow the procedure described in "To Perform Authentication using a One-Time Password" to verify that you can use the ForgeRock Authenticator app to perform multi-factor authentication. If the authentication tree is correctly configured, authentication is successful and AM displays the user profile page, without having to enter a password.

Creating Multi-Factor Authentication Chains

In AM 6 it is recommended that you implement multi-factor authentication using authentication trees. For more information, see "Creating Authentication Trees for Push Authentication".

The following sections provide steps for creating authentication chains that implement multi-factor authentication.

Creating Authentication Chains for Push Authentication

Push authentication uses two separate authentication modules:

- A module to register a device to receive push notifications called *ForgeRock Authenticator (Push) Registration*.
- A module to perform the actual authentication itself, called *ForgeRock Authenticator (Push)*.

You can insert both modules into a single chain to register devices and then authenticate with push notifications. See "To Create an Authentication Chain for Push Authentication".

The ForgeRock Authenticator (Push) module can also be used for passwordless authentication using push notifications. If the module is placed at the start of a chain, it will ask the user to enter their user ID, but not their password. A push notification is then sent to their registered device to complete the authentication by using the ForgeRock Authenticator app.

For information on configuring an authentication chain for passwordless authentication, see "To Create an Authentication Chain for Push Registration and Passwordless Authentication".

For information on the potential limitations of passwordless authentication, see "Limitations When Using Passwordless Push Authentication".

To Create an Authentication Chain for Push Authentication

The procedure assumes the following:

- Users will provide user IDs and passwords as the first step of multi-factor authentication.
- If the user does not have a device registered to receive push notifications, they will be asked to register a device. After successfully registering a device for push, authentication will proceed to the next step.
- A push notification will be sent to the device as a second factor to complete authentication.

To create a multi-factor authentication chain that uses the ForgeRock Authenticator (Push) Registration and ForgeRock Authenticator (Push) modules, perform the following steps:

1. Log in to the AM console as an AM administrator, for example `amadmin`.
2. Select the realm that will contain the authentication chain.
3. Create a ForgeRock Authenticator (Push) Registration authentication module as follows:
    a. Select Authentication > Modules, and then click Add Module.

       The New Module page appears.

    b. Fill in fields in the New Module page as follows:

       - Name: Specify a module name of your choosing, for example *push-reg*.
       - Type: Select ForgeRock Authenticator (Push) Registration.

    c. Click Create.

       A page that lets you configure the authentication module appears.

    d. Configure the module to meet your organization's requirements.

       For more information about the authentication module's configuration settings, see "ForgeRock Authenticator (Push) Registration Authentication Module".
4. Create a ForgeRock Authenticator (Push) authentication module as follows:

   a. Select Authentication > Modules, and then click Add Module.

      The New Module page appears.

   b. Fill in fields in the New Module page as follows:

      - Name: Specify a module name of your choosing, for example *push-authn*.

      - Type: Select ForgeRock Authenticator (Push).

   c. Click Create.

      A page that lets you configure the authentication module appears.

   d. Configure the module to meet your organization's requirements.

      For more information about the authentication module's configuration settings, see "ForgeRock Authenticator (Push) Authentication Module".

5. Create the authentication chain as follows:

   a. Select Authentication > Chains, and then click Add Chain.

      The Add Chain page appears.

   b. Specify a name of your choosing, for example `myPushAuthChain`, and then click Create.

      A page appears with the Edit Chain tab selected.

   c. Add the Data Store authentication module to the authentication chain as follows:

      i. Click Add a Module.

         The New Module dialog box appears.

      ii. Fill in the New Module dialog box, specifying the Data Store authentication module. For this example, specify the `Requisite` flag.

      iii. Click OK.

         The graphic showing your authentication chain now includes a Data Store authentication module.

   d. Add the ForgeRock Authenticator (Push) Registration authentication module to the authentication chain as follows:

      i. Click Add a Module.

         The New Module dialog box appears.

ii. Fill in the New Module dialog box, specifying the ForgeRock Authenticator (Push) Registration authentication module that you just created. For this example, specify the `Requisite` flag.

iii. Click OK.

The graphic showing your authentication chain now includes a Data Store, and a ForgeRock Authenticator (Push) Registration authentication module.

e. Add the ForgeRock Authenticator (Push) authentication module to the authentication chain as follows:

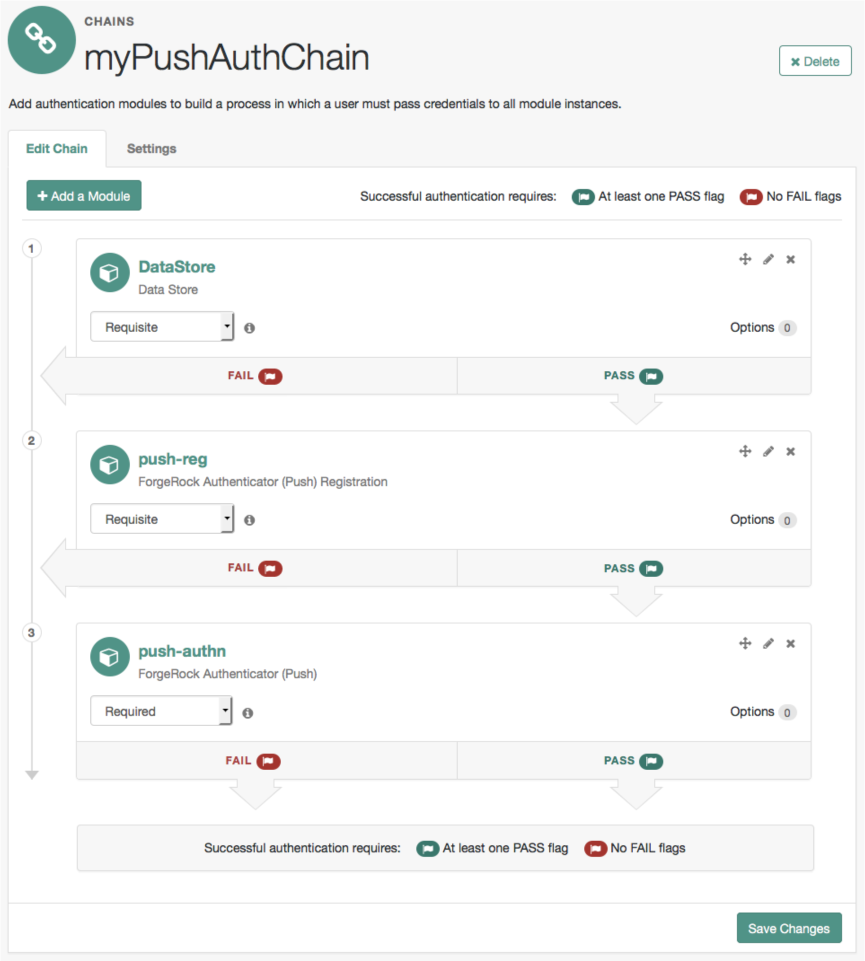
i. Click Add a Module.

The New Module dialog box appears.

ii. Fill in the New Module dialog box, specifying the ForgeRock Authenticator (Push) authentication module that you created. For this example, specify the `Required` flag.

iii. Click OK.

The graphic showing your authentication chain now includes a Data Store, a ForgeRock Authenticator (Push) Registration, and a ForgeRock Authenticator (Push) authentication module.



CHAINS

myPushAuthChain

Add authentication modules to build a process in which a user must pass credentials to all module instances.

Successful authentication requires: At least one PASS flag, No FAIL flags

1. DataStore (Data Store) — Requisite — Options 0
2. push-reg (ForgeRock Authenticator (Push) Registration) — Requisite — Options 0
3. push-authn (ForgeRock Authenticator (Push)) — Required — Options 0

Successful authentication requires: At least one PASS flag, No FAIL flags

f. Save your changes.

6. Test your authentication chain as follows:

   a. Logout of AM, and then navigate to a URL similar to the following: `http://openam.example.com:8443/openam/XUI/?realm=/&service=`*`myPushAuthChain`*`#login`

      A login screen prompting you to enter your user ID and password appears.

   b. Follow the procedure described in "To Perform Authentication using Push Notifications" to verify that you can use the ForgeRock Authenticator app to perform multi-factor authentication. If the chain is correctly configured, authentication is successful and AM displays the user profile page.

To Create an Authentication Chain for Push Registration and Passwordless Authentication

The procedure assumes the following:

- Users will provide only their user IDs as the first step of multi-factor authentication.
- The user already has a device registered for receiving push notifications. For details of an authentication chain which can register a device for push notifications, see "To Create an Authentication Chain for Push Authentication".
- A push notification will be sent to the device as a second factor, to complete authentication without the need to enter a password.

To create a multi-factor authentication chain that uses the ForgeRock Authenticator (Push) module for passwordless authentication, perform the following steps:

1. Log in to the AM console as an AM administrator, for example `amadmin`.
2. Select the realm that will contain the authentication chain.
3. Create the authentication chain as follows:
   a. Select Authentication > Chains, and then click Add Chain.

      The Add Chain page appears.

   b. Specify a name of your choosing, for example *myPasswordlessAuthChain*, and then click Create.

      A page appears with the Edit Chain tab selected.

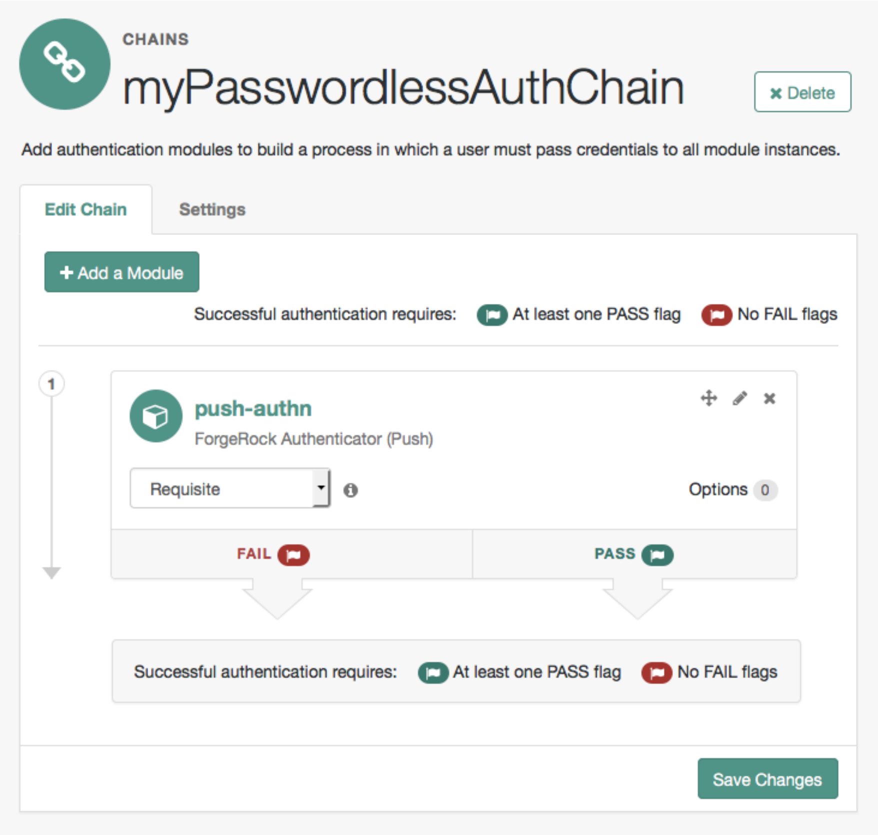
   c. Add the ForgeRock Authenticator (Push) authentication module to the authentication chain as follows:
      i. Click Add a Module.

         The New Module dialog box appears.

      ii. Fill in the New Module dialog box, specifying the ForgeRock Authenticator (Push) authentication module that you created. For this example, specify the `Requisite` flag.

      iii. Click OK.

         The graphic showing your authentication chain now includes a ForgeRock Authenticator (Push) authentication module.



CHAINS

myPasswordlessAuthChain

Add authentication modules to build a process in which a user must pass credentials to all module instances.

Successful authentication requires: At least one PASS flag No FAIL flags

1

push-authn

ForgeRock Authenticator (Push)

Requisite

Options 0

Successful authentication requires: At least one PASS flag No FAIL flags

d. Save your changes.

4. Test your authentication chain as follows:

   a. Logout of AM, and then navigate to a URL similar to the following: `http://openam.example.com:8443/openam/XUI/?realm=/#login/&service=myPasswordlessAuthChain`

      A login screen prompting you to enter your user ID appears.

   b. Follow the procedure described in "To Perform Authentication using Push Notifications" to verify that you can use the ForgeRock Authenticator app to perform multi-factor authentication. If the chain is correctly configured, authentication is successful and AM displays the user profile page, without having to enter a password.

Creating Authentication Chains for One-Time Password Authentication

This section covers one-time password authentication.

To Create an Authentication Chain for One-Time Password Authentication

The procedure assumes the following:

- Users will provide user IDs and passwords as the first step of multi-factor authentication.
- An existing Data Store authentication module will collect and verify user IDs and passwords.
- All authentication modules in the chain will use the `Requisite` flag setting. See "About Authentication Modules and Chains" for details about authentication module flag settings.
- Users can opt out of one-time password authentication.

To create a multi-factor authentication chain that uses the ForgeRock Authenticator (OATH) module, perform the following steps:

1. Log in to the AM console as an AM administrator, for example `amadmin`.
2. Select the realm that will contain the authentication chain.
3. You can allow users to opt out of using OATH-based one-time passwords as follows:
   a. Select Authentication > Settings > General.
   b. Make sure that the Two Factor Authentication Mandatory is not enabled.

      See "General" for details about this configuration setting.

   For information about how letting users skip multi-factor authentication impacts the behavior of authentication chains, see "Letting Users Opt Out of One-Time Password Authentication".
4. Create a ForgeRock Authenticator (OATH) authentication module as follows:
   a. Select Authentication > Modules, and then click Add Module.

      The New Module page appears.
   b. Fill in fields in the New Module page as follows:
      - Name: Specify a module name of your choosing.
      - Type: Select ForgeRock Authenticator (OATH).
   c. Click Create.

      A page that lets you configure the authentication module appears.

d. Configure the ForgeRock Authenticator authentication module to meet your organization's requirements.

   For more information about the authentication module's configuration settings, see "ForgeRock Authenticator (OATH) Authentication Module".

5. Create the authentication chain as follows:

   a. Select Authentication > Chains, and then click Add Chain.

      The Add Chain page appears.

   b. Specify a name of your choosing, for example *myOATHAuthChain*, and then click Create.

      A page appears with the Edit Chain tab selected.

   c. Click Add a Module. Fill in fields in the New Module dialog box as follows:

      - Select Module: Select the existing Data Store module to use in this chain.
      - Select Criteria: Select a flag setting for the module in the authentication chain. For this example, specify the `Requisite` flag.

        See "About Authentication Modules and Chains" for information about authentication module flag settings.

   d. Click OK.

      A graphic showing an authentication chain with a single Data Store module appears on the page.

   e. Add the ForgeRock Authenticator (OATH) authentication module to the authentication chain as follows:

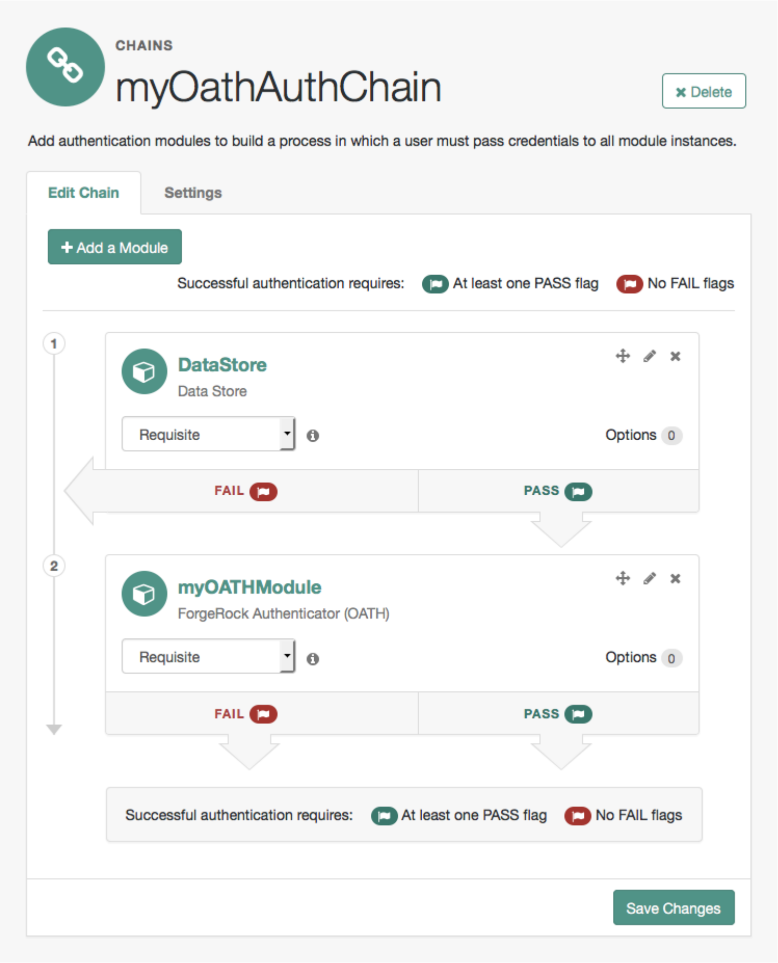
      i. Click Add a Module.

         The New Module dialog box appears.

      ii. Fill in the New Module dialog box, specifying the ForgeRock Authenticator (OATH) authentication module that you just created. For this example, specify the `Requisite` flag.

      iii. Click OK.

         The graphic showing your authentication chain now includes the Data Store and ForgeRock Authenticator (OATH) authentication module.



CHAINS

myOathAuthChain

Add authentication modules to build a process in which a user must pass credentials to all module instances.

Successful authentication requires: At least one PASS flag No FAIL flags

1 DataStore — Data Store — Requisite — Options 0

2 myOATHModule — ForgeRock Authenticator (OATH) — Requisite — Options 0

Successful authentication requires: At least one PASS flag No FAIL flags

f. Save your changes.

6. Test your authentication chain as follows:

   a. Logout of AM, and then navigate to a URL similar to the following: `http://openam.example.com:8443/openam/XUI/?realm=/&service=myOATHAuthChain#login`

      A login screen prompting you to enter your user ID and password appears.

   b. Follow the procedure described in "To Perform Authentication using a One-Time Password" to verify that you can use the ForgeRock Authenticator app to perform multi-factor authentication. If the chain is correctly configured, authentication is successful and AM displays the user profile page.

Managing Devices for Multi-Factor Authentication

Multi-factor authentication requires you to register a device, which is used as an additional factor when you log in to AM.

This section covers the following topics relating to devices used for multi-factor authentication:

- "Downloading the ForgeRock Authenticator App"
- "Registering the ForgeRock Authenticator for Multi-Factor Authentication"
- "Opting Out of One-Time Password Authentication"
- "Recovering After Replacing a Lost Device"
- "Recovering After a Device Becomes Out of Sync"
- "Resetting Registered Devices by using REST"

Downloading the ForgeRock Authenticator App

If you have not already done so, download and install the ForgeRock Authenticator app on your phone, so that you can perform multi-factor authentication.

The ForgeRock Authenticator app supports push authentication notifications and one-time passwords.

The app is available for both Android and iOS devices, and is free to download. Source code is also available:

Android

Download: Google Play

Source code: https://stash.forgerock.org/projects/OPENAM/repos/forgerock-authenticator-android

iOS

Download: App Store

Source code: https://stash.forgerock.org/projects/OPENAM/repos/forgerock-authenticator-ios

Registering the ForgeRock Authenticator for Multi-Factor Authentication

Registering the ForgeRock Authenticator app enables it to be used as an additional factor when logging in to AM.

The ForgeRock Authenticator app supports registration of multiple accounts and multiple different authentication methods in each account, such as push notifications and one-time passwords.

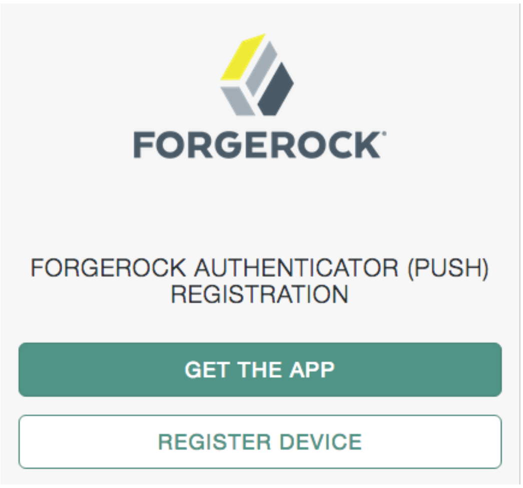
For information on registering Web Authentication (WebAuthn) devices with AM, see "Creating Trees for Web Authentication (WebAuthn)".

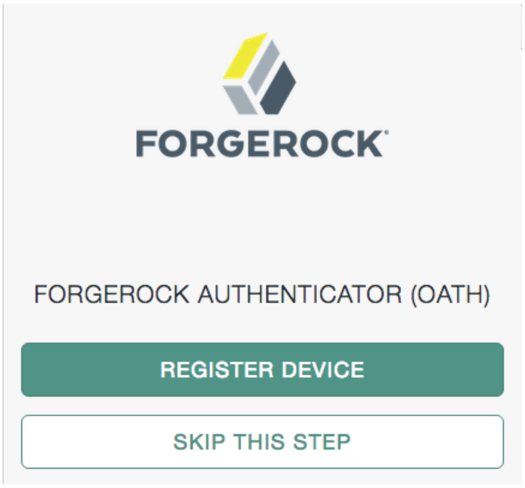
ForgeRock Authenticator registration only needs to be completed the first time an authentication method is used with an identity provider. Use of a different authentication method may require that registration with the identity provider is repeated for that additional method.

The ForgeRock Authenticator needs access to the internet to register to receive push notifications. Registering for one-time password authentication does not require a connection to the internet.

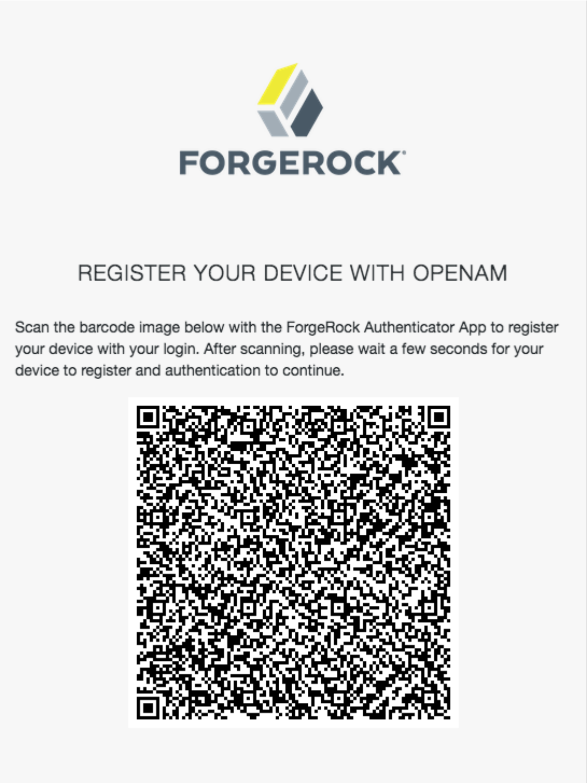
To Register the ForgeRock Authenticator for Multi-Factor Authentication

1. When visiting a protected resource without having any registered devices for multi-factor authentication, AM requires that you register a device.



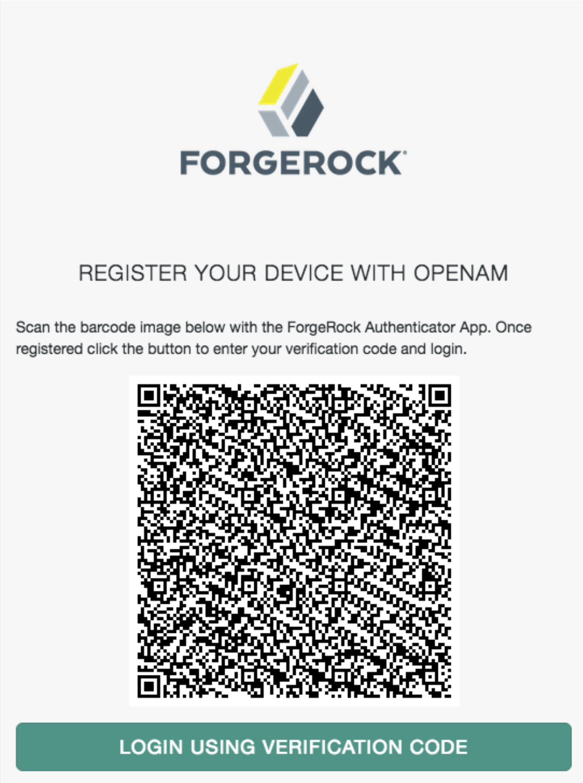


To register your mobile phone with AM, click Register Device. A screen with a QR code appears:



REGISTER YOUR DEVICE WITH OPENAM

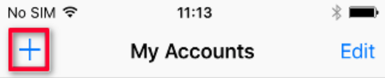
Scan the barcode image below with the ForgeRock Authenticator App to register your device with your login. After scanning, please wait a few seconds for your device to register and authentication to continue.



REGISTER YOUR DEVICE WITH OPENAM

Scan the barcode image below with the ForgeRock Authenticator App. Once registered click the button to enter your verification code and login.

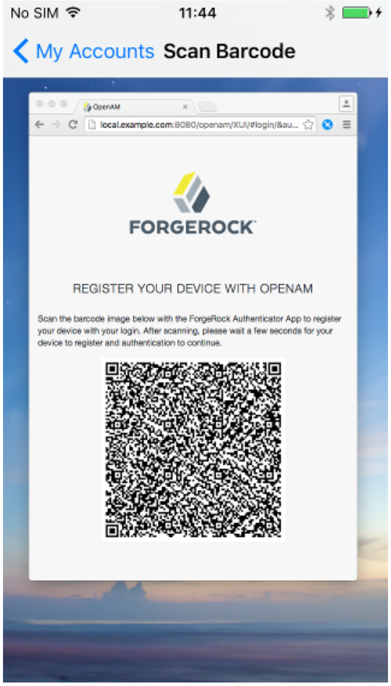
2. Start the ForgeRock Authenticator app on the device to register, and then click the plus icon:



The screen on the device changes to an interface similar to your camera app.

3. Point the camera at the QR code on the AM page and the ForgeRock Authenticator app will acquire the QR code and read the data encoded within.

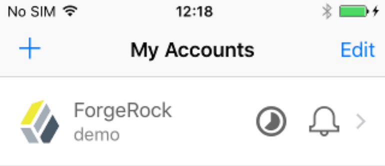
If you are logging in to AM on the registered device and cannot scan the screen, click the button labelled On a mobile device?. The ForgeRock Authenticator app will request permission to launch. If allowed, the information required to register the device will be transferred to the ForgeRock Authenticator app directly, without the need to scan the QR code.



REGISTER YOUR DEVICE WITH OPENAM

Scan the barcode image below with the ForgeRock Authenticator App to register your device with your login. After scanning, please wait a few seconds for your device to register and authentication to continue.

4. Once registered, the app displays the registered accounts and the authentication methods they support, for example one-time passwords (a timer icon) or push notifications (a bell icon):



5. When registering a device, you MUST make a copy of the recovery codes associated with that device.

   Depending on the device type you registered, perform one of the following steps:

   - If you registered an OATH device:

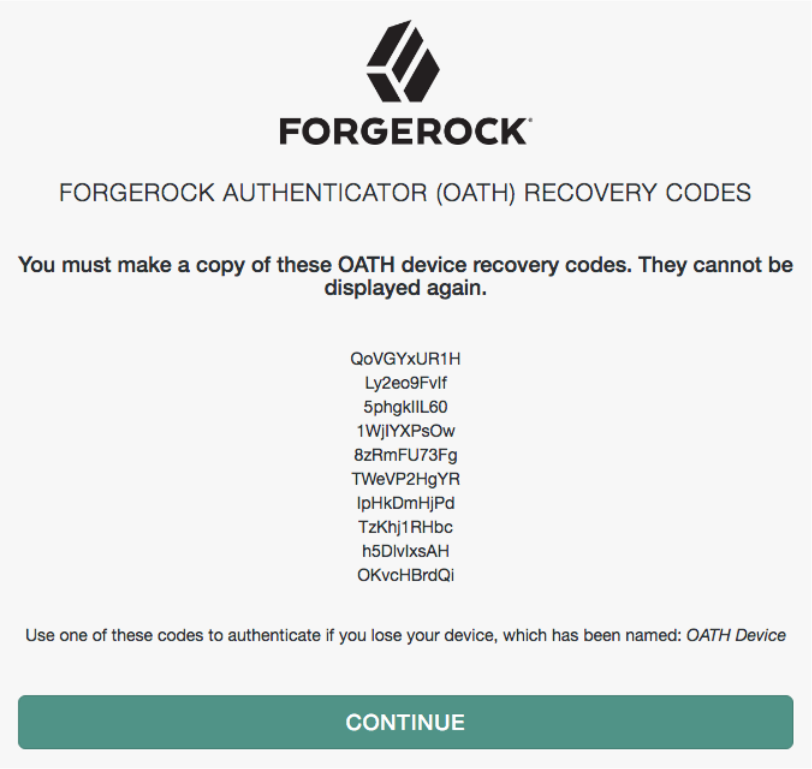
     a. Click the Login Using Verification Code button.

        You will be asked to enter a verification code.

     b. In the ForgeRock Authenticator app, click the newly registered account, and then click the Refresh button to generate a new one-time password.

c. Enter the one-time password into the web page, and then click Submit.

d. On the recovery codes page, make a copy of the displayed recovery codes and store them safely. The codes will never be displayed again.



FORGEROCK AUTHENTICATOR (OATH) RECOVERY CODES

You must make a copy of these OATH device recovery codes. They cannot be displayed again.

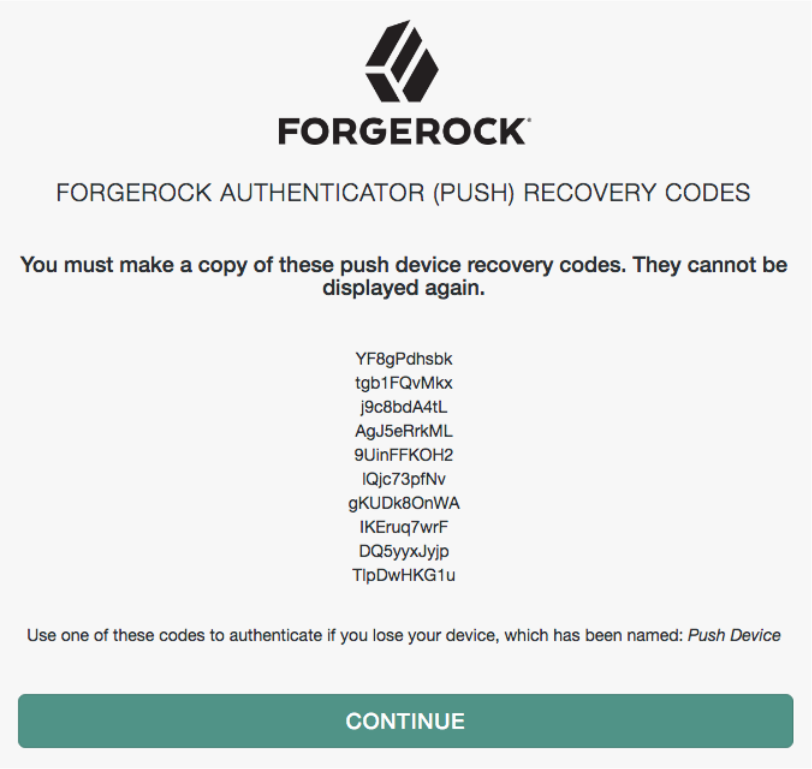
QoVGYxUR1H
Ly2eo9FvIf
5phgkIIL60
1WjIYXPsOw
8zRmFU73Fg
TWeVP2HgYR
IpHkDmHjPd
TzKhj1RHbc
h5DlvlxsAH
OKvcHBrdQi

Use one of these codes to authenticate if you lose your device, which has been named: *OATH Device*

When you have safely stored the recovery codes for your newly registered OATH device, click the Continue button.

- If you registered a push device:

  - On the recovery codes page, make a copy of the displayed recovery codes and store them safely. The codes will never be displayed again.



FORGEROCK

FORGEROCK AUTHENTICATOR (PUSH) RECOVERY CODES

You must make a copy of these push device recovery codes. They cannot be displayed again.

```
YF8gPdhsbk
tgb1FQvMkx
j9c8bdA4tL
AgJ5eRrkML
9UinFFKOH2
lQjc73pfNv
gKUDk8OnWA
IKEruq7wrF
DQ5yyxJyjp
TlpDwHKG1u
```

Use one of these codes to authenticate if you lose your device, which has been named: *Push Device*

When you have safely stored the recovery codes for your newly registered push device, click the Continue button.

Your device is now registered. You will able to use it to perform multi-factor authentication.

Opting Out of One-Time Password Authentication

Unless the AM administrator has made one-time password authentication mandatory, users can choose to opt out of using one-time passwords by clicking the Skip This Step button on the ForgeRock Authenticator (OATH) screen. [1] This button appears:

- When users are prompted to register their mobile devices during their initial login from a new device.

- Every time users are prompted by the ForgeRock Authenticator (OATH) authentication module to enter one-time passwords.

[1] For information about making the usage of one-time passwords mandatory in AM, see "Letting Users Opt Out of One-Time Password Authentication".

Users who decide to opt out of using one-time passwords are not prompted to enter one-time passwords when authenticating to AM.

The decision to opt out of using one-time passwords in AM is revocable: users who have decided to opt out of using one-time passwords can reverse their decisions, so that one-time password authentication is once again required.

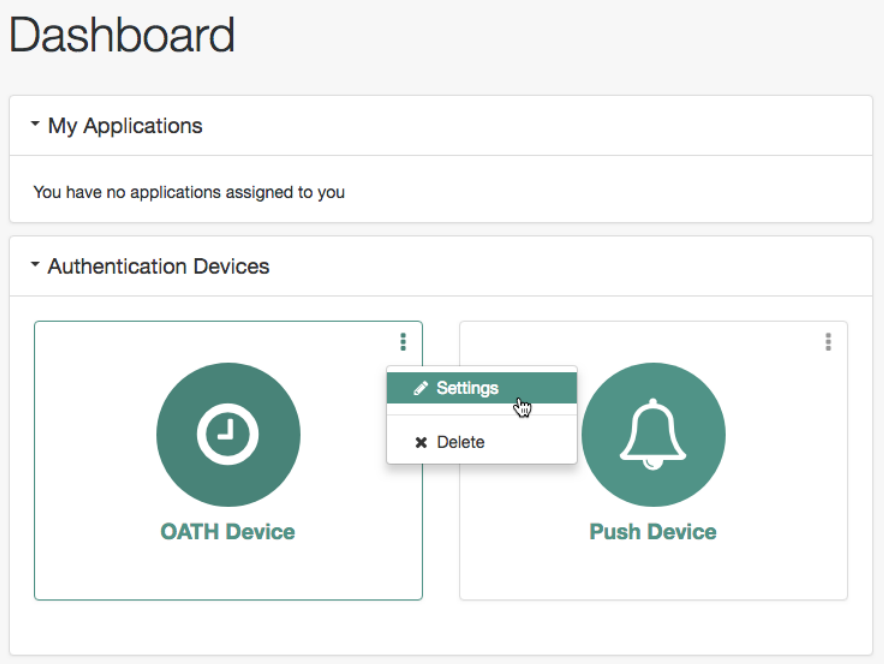
End users should follow these steps to opt out or opt in to using one-time passwords:

To Opt out or Opt in to Using One-Time Passwords

1. Log in to AM.

2. Select Dashboard from the top navigation bar.

3. In the Authentication Devices section of the Dashboard page, click the context menu button for the chosen device, and then click Settings:



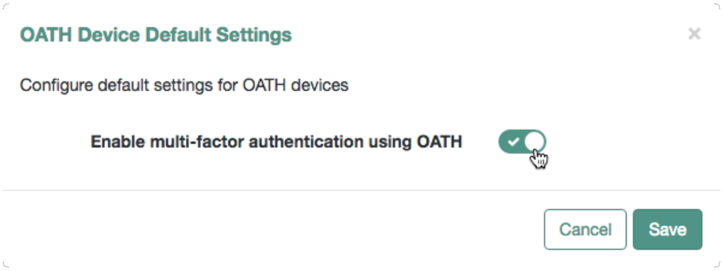
Dashboard

My Applications

You have no applications assigned to you

Authentication Devices

4. Enable or disable the multi-factor authentication option:



OATH Device Default Settings

Configure default settings for OATH devices

Enable multi-factor authentication using OATH ✓

5. Click Save.

Recovering After Replacing a Lost Device

If you register a device with AM and then lose it, you must authenticate to AM using a recovery code, delete the lost device, and then register the new device. Perform the following steps:

To Register a New Device After Losing a Registered Device

1. Log in to AM. If push authentication is enabled, enter your user ID, click Log In, and then click Use Emergency Code. If one-time passwords are enabled, when prompted to enter a verification code, instead enter one of your recovery codes.

   Because recovery codes are valid for a single use only, make a note to yourself not to attempt to reuse this code.

   If you did not save the recovery codes for the lost device, contact your administrator to remove the registered device from your AM user profile.

2. Select Dashboard from the top-level menu.

3. Locate the entry for your phone in the Authentication Devices section, click the context menu button, and then click Delete.

4. If you have not already done so, install the ForgeRock Authenticator app on your new phone. See "Downloading the ForgeRock Authenticator App".

5. Register your new device. See "Registering the ForgeRock Authenticator for Multi-Factor Authentication".

Users who do not save recovery codes or who run out of recovery codes and cannot authenticate to AM without a verification code require administrative support to reset their device profiles. See "Resetting Registered Devices by using REST" for more information.

Recovering After a Device Becomes Out of Sync

If you repeatedly enter valid one-time passwords that appear to be valid passwords, but AM rejects the passwords as unauthorized, it is likely that your device has become out of sync with AM.

When a registered device becomes out of sync with AM, you must authenticate to AM using a recovery code, delete your device, and then re-register your device. You can do so by performing the steps in "To Register a New Device After Losing a Registered Device".

Users who do not save recovery codes or who run out of recovery codes and cannot authenticate to AM without a verification code require administrative support to reset their device profiles. See "Resetting Registered Devices by using REST" for more information.

Resetting Registered Devices by using REST

As described in "Recovering After Replacing a Lost Device", a user who has lost a mobile phone registered with AM can register a replacement device by authenticating using a recovery code, deleting their existing device, and then re-registering a new device.

Additional support is required for users who lose mobile phones but did not save their recovery codes when they initially registered the phone, and for users who have used up all their recovery codes.

AM provides a REST API to reset a device profile by deleting information about a user's registered device. Either the user or an administrator can call the REST API to reset a device profile. Device profile reset can be implemented as follows:

- Administrators provide authenticated users with a self-service page that calls the REST API to let the users reset their own device profiles.
- Administrators can call the REST API themselves to reset users' device profiles.
- Administrators can call the REST API themselves to reset a device when the HOTP counter exceeds the HOTP threshold window and requires a reset.

> **Note**
>
> The reset action deletes the OATH device profile, which by default has a limit of one profile per device, and sets the `Select to Enable Skip` option to its default value of `Not Set`.

Reset OATH Devices

To reset a user's OATH device profile, perform an HTTP POST to the `/users/user/devices/2fa/oath?_action=reset` endpoint.

When making a REST API call, specify the realm in the path component of the endpoint. You must specify the entire hierarchy of the realm, starting at the top-level realm. Prefix each realm in the hierarchy with the `realms/` keyword. For example `/realms/root/realms/customers/realms/europe`.

The following example resets the OATH devices of a user named `myUser` in a subrealm of the top-level realm called `mySubrealm`:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.0" \
--header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
--data '{}' \
https://openam.example.com:8443/openam/json/realms/root/realms/mySubrealm/users/myUser/devices/2fa/oath?
_action=reset
{
    "result":true
}
```

Reset Push Devices

To reset push devices over REST, perform an HTTP POST to the `/users/user/devices/2fa/push?_action=reset` endpoint as follows:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.0" \
--header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
--data '{}' \
https://openam.example.com:8443/openam/json/realms/root/realms/mySubrealm/users/myUSER/devices/2fa/push?
_action=reset"
{
    "result":true
}
```

Authenticating Using Multi-Factor Authentication

This section provides an example of how end users might authenticate with AM configured for multi-factor authentication. Use the following procedures to complete multi-factor authentication using the ForgeRock Authenticator:

- "To Perform Authentication using a One-Time Password"
- "To Perform Authentication using Push Notifications"

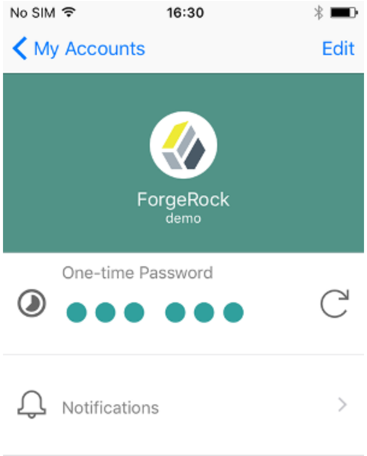
To Perform Authentication using a One-Time Password

This example uses the authentication chain as created in "Creating Authentication Chains for One-Time Password Authentication".

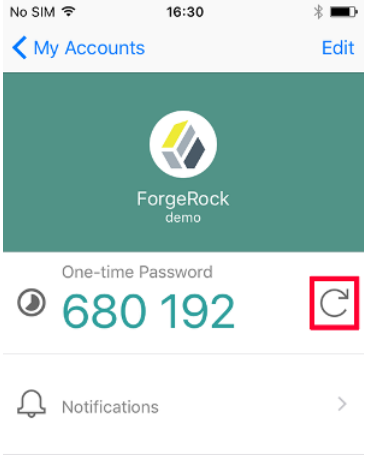
Because the first module in the authentication chain is a Data Store module, AM presents you with a page for entering your user ID and password. After you provide those credentials, AM verifies them. If your credentials are valid, AM proceeds to the ForgeRock Authenticator (OATH) authentication module.

On the ForgeRock Authenticator (OATH) screen, follow these steps to complete one-time password authentication:

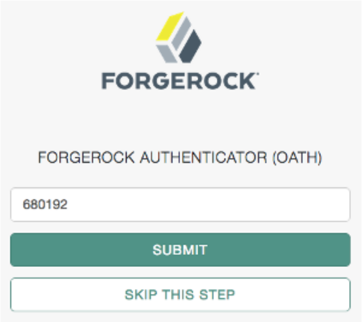
1. On your registered device, open the ForgeRock Authenticator app, and then tap the account matching the user ID you entered earlier. The registered authentication methods for that account are displayed:



2. In the One-time Password section, click the refresh icon. A one-time password is displayed:



3. On the ForgeRock Authenticator (OATH) page in AM, enter the one-time password that the authenticator app generated on your phone, and then click Submit:



FORGEROCK

FORGEROCK AUTHENTICATOR (OATH)

680192

AM will display the user's profile page.

To Perform Authentication using Push Notifications

This example uses one of the authentication chains as created in "Creating Authentication Chains for Push Authentication".

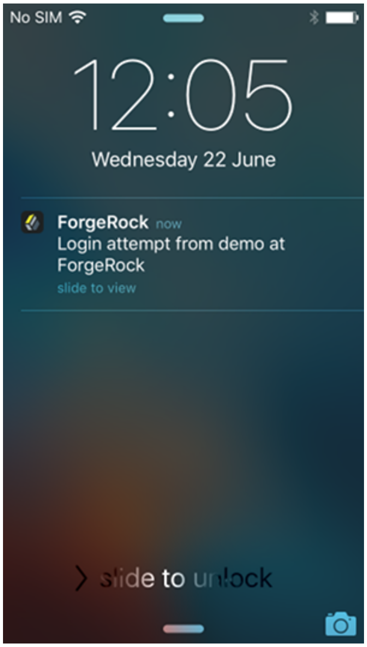
AM presents you with a page for entering only your user ID, or user ID and password. After you provide those credentials, AM verifies them. If your credentials are valid and the account has a device registered for push notifications, AM proceeds to the ForgeRock Authenticator (Push) authentication module, and a push notification is sent to the registered device.

> **Note**
>
> The device needs access to the Internet to receive push notifications, and the AM server must be able to receive responses from the device.

Follow these steps to complete authentication using push notifications:

1. On your registered device, you will receive a push notification from AM. Depending on the state of the phone and the ForgeRock Authenticator app, respond to the notification as follows:

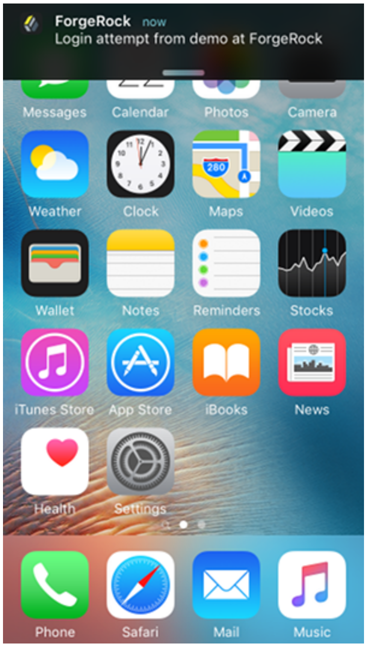
    - If the phone is locked, the notification may appear similar to the following:



ForgeRock now
Login attempt from demo at ForgeRock
slide to view

Slide the notification across the screen, then unlock the phone. The ForgeRock Authenticator app will automatically open and display the push notification authentication screen.

- If the phone is not locked, and the ForgeRock Authenticator app is not open, the notification may appear similar to the following:



ForgeRock now
Login attempt from demo at ForgeRock

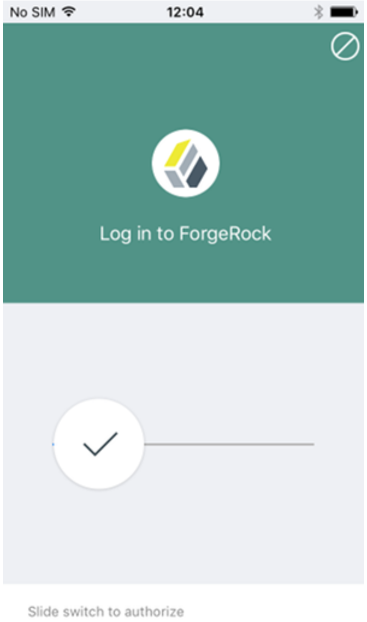
Tap the notification. The ForgeRock Authenticator app will automatically open and display the push notification authentication screen.

- If the phone is not locked, and the ForgeRock Authenticator app is open, the app will open the push notification authentication screen automatically.

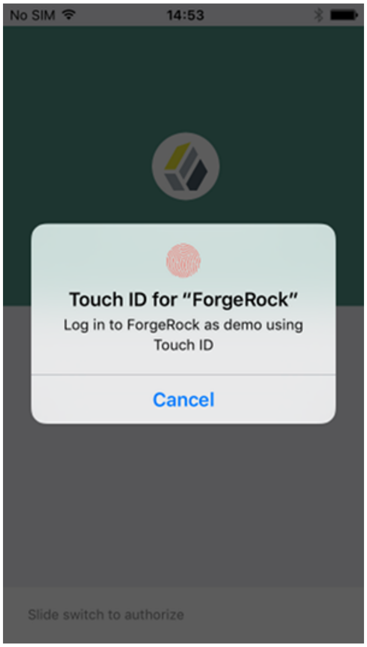
On the push notification authentication screen, you can approve the request or deny it.

2. (Optional) Approve the request using one of the following methods:

   - Slide the switch with a checkmark on horizontally to the right.



- If the registered device supports Touch ID, and fingerprints have been provided, you can approve the request by using a registered fingerprint.



Touch ID for “ForgeRock”

Log in to ForgeRock as demo using Touch ID

- If the registered device supports face recognition and you have set up facial recognition, you can approve the request by glancing at your device.

  AM will display the user's profile page.

- (Optional) Deny the request by tapping the cancel icon in the top-right of the screen or, if Touch ID or face recognition are enabled, tap the Cancel button.

  After a timeout has passed, AM will report that authentication has failed and return to the first screen in the chain.

Note

If you do not approve or deny the request on the registered device, the AM Push Authentication page will timeout and the authentication will fail. The timeout can be configured in the ForgeRock Authenticator (Push) authentication module settings. See "ForgeRock Authenticator (Push) Authentication Module".

Chapter 5
Implementing Account Lockout

Account lockout is a security mechanism that locks a user after repeated failed login attempts.

Most deployments implement the backend user store's password policy to control account lockout. If that is not an option to your deployment, you can configure account lockout.

> **Note**
>
> Account lockout settings only apply to authentication modules and chains.
>
> To implement account lockout in authentication trees, use an Account lockout Node together with a Retry Limit Decision Node.

AM supports two different approaches to *account lockout*, where AM locks an account after repeated authentication failures—persistent lockout and memory lockout:

- Persistent (physical) lockout sets the user account status to `inactive` in the user profile. For persistent lockout, AM tracks failed authentication attempts by writing to the user repository.

  Persistent account lockout works independently of account lockout mechanisms in the underlying directory server that serves as the user data store.

- Memory lockout locks the user account, keeping track of the locked state only in memory, and then unlocking the account after a specified delay. Memory lockout is also released when AM restarts.

> **Note**
>
> Failed login attempts during the transactional authorization flow do not increment account lockout counters. For more information on transactional authorization, see "*Implementing Transactional Authorization*" in the *Authorization Guide*.
>
> If login failures are stored in AM's memory, this may result in user accounts not being locked out even after multiple login failures. To avoid this issue, make sure to implement persistent lockout instead. For more information, see "Configuring Account Lockout".

This chapter describes how to configure account lockout in AM.

Configuring Account Lockout

You configure account lockout by editing settings for the core authentication module. For details, see "Setting up a Realm for Authentication".

To Configure Account Lockout

1. Access the settings in the AM console under Realms > *Realm Name* > Authentication > Settings > Account Lockout.
2. Enable lockout by checking Login Failure Lockout Mode, setting the number of attempts, and setting the lockout interval and duration.

   You can also opt to warn users after several consecutive failures, or to multiply the lockout duration on each successive lockout.
3. (Optional) If you have configured CTS-based or client-based authentication sessions, ensure the Store Invalid Attempts in Data Store switch is enabled. Failure to do so may result in users not being locked out even after multiple login failures.
4. To save account login failures to the Data Store, enable Store Invalid Attempts in Data Store. This step is necessary when using CTS-based or client-based authentication sessions.
5. You can set up email notification upon lockout to an administrator if AM is configured to send mail. You can configure AM to send mail in Configure > Server Defaults > General > Mail Server.
6. For persistent lockout, AM sets the value of the user's `inetuserstatus` profile attribute to `inactive`. You can also specify another attribute to update on lockout. You can further set a non-default attribute on which to store the number of failed authentication attempts. When you do store the number of failed attempts in the data store, other AM servers accessing the user data store can also see the number.

> **Note**
>
> To unlock a user's account, find the user under Realms > *Realm Name* > Identities. Select the user you want to unlock, and set the user's User Status to Active. Save your changes.

Chapter 6

Implementing Sessions

Before you configure sessions in your environment, ensure you are familiar with the differences between sessions and authentication sessions, and where they are stored. For more information, see "About Sessions".

You can configure authentication session storage location independently from session storage location. For example, you could configure the same realm for client-based authentication sessions and CTS-based sessions if it suits your environment.

To configure sessions, see the following sections:

- "Implementing CTS-Based Sessions"
- "Implementing Client-Based Sessions"
- "Implementing In-Memory Authentication Sessions"

To configure authentication session whitelisting, a feature that protects authentication sessions from replay attacks, see "Implementing Authentication Session Whitelisting".

Implementing CTS-Based Sessions

By default, AM configures the CTS token store schema in the AM configuration store. Before configuring your AM deployment to use CTS-based sessions or authentication sessions, we recommend you install and configure an external CTS token store. For more information, see "*Implementing the Core Token Service*" in the *Installation Guide*.

CTS-based sessions and authentication sessions benefit from configuring sticky load balancing. For more information, see "Configuring Load Balancing for a Site" in the *Installation Guide*.

To configure CTS-based sessions and authentication sessions, see the following procedures:

- To Configure CTS-Based Authentication Sessions

- To Configure CTS-Based Sessions

To configure session quotas, a feature that lets you limit the amount of active sessions for a user, see "Implementing Session Quotas".

To Configure CTS-Based Authentication Sessions

> **Important**
>
> Configuring storage location for authentication sessions is only supported for authentication trees. Authentication chains always store authentication sessions in AM's memory. For more information, see "Session Storage Location".

1. Log in to the AM console as an administrative user, for example, `amAdmin`.
2. Navigate to Realms > *Realm Name* > Authentication > Settings > Trees.
3. From the Authentication session state management scheme drop-down list, select `CTS`.
4. In the Max duration (minutes) field, enter the maximum life of the authentication session in minutes.
5. Save your changes.
6. Navigate to Configure > Authentication > Core > Security.
7. In the Organization Authentication Signing Secret field, enter a base64-encoded HMAC secret that AM uses to sign the JWT that is passed back and forth between the client and AM during the authentication process. The secret must be, at least, 128-bits in length.
8. Save your changes.

To Configure CTS-Based Sessions

1. Log in to the AM console as an administrative user, for example, `amAdmin`.
2. Navigate to Realms > *Realm Name* > Authentication > Settings > General.
3. Ensure the Use Client-based Sessions check box is not selected.
4. Save your changes.
5. Verify that AM creates a CTS-based session when non-administrative users authenticate to the realm. Perform the following steps:
   1. Authenticate to AM as a non-administrative user in the realm you enabled for CTS-based sessions.
   2. In a different browser, authenticate to AM as an administrative user, for example, `amAdmin`.
   3. Navigate to Realms > *Realm Name* > Sessions.
   4. Verify that a session is present for the non-administrative user.

Implementing Session Quotas

AM lets you limit the number of active sessions for a user by setting session quotas. You also configure session quota exhaustion actions so that when a user goes beyond the session quota, AM takes the appropriate action.

AM's support for session quotas requires CTS-based sessions.

To Configure Session Quotas and Exhaustion Actions

The session quota applies to all sessions opened for the same user (as represented by the user's universal identifier). To configure:

1. Log in to the AM console as an administrative user, for example, `amAdmin`.

2. Navigate to Configure > Global Services > Sessions.

3. Set Enable Quota Constraints to `ON`.

4. Set Resulting behavior if session quota exhausted.

   The following settings are available by default:

   `DENY_ACCESS`

   Deny access, preventing the user from creating an additional session.

   `DESTROY_NEXT_EXPIRING`

   Remove the next session to expire, and create a new session for the user. The next session to expire is the session with the minimum time left until expiration.

   This is the default setting.

   `DESTROY_OLDEST_SESSION`

   Remove the oldest session, and create a new session for the user.

   `DESTROY_OLD_SESSIONS`

   Remove all existing sessions, and create a new session for the user.

   If none of these session quota exhaustion actions fit your deployment, you can implement a custom session quota exhaustion action. For an example, see "Customizing CTS-Based Session Quota Exhaustion Actions".

5. Set Active User Sessions to the session quota.

   The default is 5 sessions.

6. Save your work.

Implementing Client-Based Sessions

To configure client-based sessions and authentication sessions for a realm, see the following sections:

- Planning for Client-Based Sessions
- Configuring Client-Based Sessions
- Configuring Client-Based Session and Authentication Session Security
- Configuring Session Blacklisting
- Limitations When Using Client-Based Sessions

Planning for Client-Based Sessions

Before configuring your AM deployment to use client-based sessions or client-based authentication sessions, perform the following tasks:

- Ensure the trust store used by AM has the necessary certificates installed:
  - A certificate is required for encrypting JWTs containing client-based sessions.
  - If you are using RS256 signing, then a certificate is required to sign JWTs. (HMAC signing uses a shared secret.)

  The same certificates must be stored on all servers participating in an AM site.

  For more information about managing certificates for AM, see "*Configuring Secrets, Certificates, and Keys*" in the *Setup and Maintenance Guide*.
- Ensure that your users' browsers can accommodate larger session cookie sizes required by client-based sessions. For more information about session cookie sizes, see "Session Cookies".
- Ensure that the AM web container can accommodate an HTTP header that is 16K in size or greater. When using Apache Tomcat as the AM web container, configure the `server.xml` file's `maxHttpHeaderSize` property to `16384` or higher.
- Ensure that your deployment does not require any of the capabilities specified in the list of limitations that apply to client-based sessions.

Configuring Client-Based Sessions

To configure client-based sessions and authentication sessions, see the following procedures:

- To Configure Client-Based Authentication Sessions
- To Configure Client-Based Sessions

To Configure Client-Based Authentication Sessions

> **Important**
>
> Configuring storage location for authentication sessions is only supported for authentication trees. Authentication chains always store authentication sessions in AM's memory. For more information, see "Session Storage Location".

1. Log in to the AM console as an administrative user, for example, `amAdmin`.
2. Navigate to Realms > *Realm Name* > Authentication > Settings > Trees.
3. From the Authentication session state management scheme drop-down list, select `JWT`.
4. In the Max duration (minutes) field, enter the maximum life of the authentication session in minutes.
5. Save your changes.
6. Navigate to Configure > Authentication > Core > Security.
7. In the Organization Authentication Signing Secret field, enter a base64-encoded HMAC secret that AM uses to sign the JWT that is passed back and forth between the client and AM during the authentication process. The secret must be, at least, 128-bits in length.
8. Save your changes.

To Configure Client-Based Sessions

1. Log in to the AM console as an administrative user, for example, `amAdmin`.
2. Navigate to Realms > *Realm Name* > Authentication > Settings > General.
3. Select the Use Client-based Sessions check box.
4. Save your changes.
5. Verify that AM creates a client-based session when non-administrative users authenticate to the realm. Perform the following steps:
   1. Authenticate to the AM console as the top-level administrator (by default, the `amAdmin` user). Note that sessions for the top-level administrator are always stored in the CTS token store.
   2. Navigate to Realms > *Realm Name* > Sessions.

c. Verify that a session is present for the `amAdmin` user.

d. In your browser, examine the AM cookie, named `iPlanetDirectoryPro` by default. Copy and paste the cookie's value into a text file and note its size.

e. Start up a private browser session that will not have access to the `iPlanetDirectoryPro` cookie for the `amAdmin` user:

   - On Chrome, open an incognito window.
   - On Internet Explorer or Microsoft Edge, start InPrivate browsing.
   - On Firefox, open a new private window.
   - On Safari, open a new private window.

f. Authenticate to AM as a non-administrative user in the realm for which you enabled client-based sessions. Be sure *not* to authenticate as the `amAdmin` user this time.

g. In your browser, examine the `iPlanetDirectoryPro` cookie. Copy and paste the cookie's value into a second text file and note its size. The size of the client-based session cookie's value should be considerably larger than the size of the cookie used by the CTS-based session for the `amAdmin` user. If the cookie is not larger, you have not enabled client-based sessions correctly.

h. Return to the original browser window in which the AM console appears.

i. Refresh the window containing the Sessions page.

j. Verify that a session still appears for the `amAdmin` user, but that no session appears for the non-administrative user in the realm with client-based sessions enabled.

Configuring Client-Based Session and Authentication Session Security

When using client-based sessions and authentication sessions, you should configure AM to sign and/or encrypt the JWT containing session information:

JWT Signing

AM verifies that the JWT is authentic by validating a signature configured in the Session Service. AM thwarts attackers who might attempt to tamper with the contents of the JWT or its signature, or who might attempt to sign the JWT with an incorrect signature.

JWT Encryption

Knowledgeable users can easily decode JWTs. Because an AM session or authentication session contains information that might be considered sensitive, encrypting the JWT that contains it protects its contents, ensuring opaqueness.

Encrypting the JWT prevents man-in-the-middle attacks that could log the state of every AM session. Encryption also ensures that end users are unable to access the information in their AM session.

Signing and encryption require certificates and shared secrets. When deploying multiple AM servers in a site, every server must have the same security configuration. Shared secrets and security keys must be identical. If you modify shared secrets or keys, you must update all the servers in the site. For more information about managing certificates for AM, see "*Configuring Secrets, Certificates, and Keys*" in the *Setup and Maintenance Guide*.

Client-based sessions and authentication sessions share the same encryption and signing configuration for the realm.

> **Important**
>
> To ensure that the client-based session cookie size does not surpass the browser supported size, Web Agents and Java Agents do not support both signing and encrypting the session cookie. For more information, see "Configure Client-Based Session Security for Agents".

The following sections provide detailed steps for configuring client-based session and authentication session cookie security:

- "Configuring the JWT Signature"
- "Configuring JWT Encryption"

Configuring the JWT Signature

Configure a JWT signature to prevent malicious tampering of client-based session and authentication session JWTs.

Perform the following steps to configure the JWT signature:

To Configure the JWT Signature

1. Navigate to Configure > Global Services > Session > Client-based Sessions.

2. From the Signing Algorithm Type drop-down list, select one of the following algorithms:

   - **HS256**. HMAC using SHA-256
   - **HS384**. HMAC using SHA-384
   - **HS512**. HMAC using SHA-512
   - **RS256**. RSASSA-PKCS1-v1_5 using SHA-256
   - **ES256**. ECDSA using SHA-256 and NIST standard P-256 elliptic curve
   - **ES384**. ECDSA using SHA-384 and NIST standard P-384 elliptic curve
   - **ES512**. ECDSA using SHA-512 and NIST standard P-521 elliptic curve

   The default value is `RS256`.

> **Note**
>
> You should disable signing the JWT if you plan to encrypt the JWT with the Direct AES Encryption algorithm. To disable JWT signing, perform the following steps:
>
> 1. Navigate to Deployment > Servers > *Server Name* > Advanced.
> 2. Set the `org.forgerock.openam.session.stateless.signing.allownone` property to `true`.
> 3. Save your changes.
> 4. Navigate to Configure > Global Services > Session > Client-based Sessions.
> 5. From the Signing Algorithm Type drop-down list, select `NONE`.
> 6. Save your changes.

3. If you specified an HMAC signing algorithm, change the value in the Signing HMAC Shared Secret field if you do not want to use the generated default value.
4. If you specified the RS256 signing algorithm, specify a value in the Encryption RSA Certificate Alias field to use for signing the JWT signature.
5. Save your changes.

For detailed information about Session Service configuration attributes, see the entries for "Session" in the *Reference*.

Configuring JWT Encryption

Configure JWT encryption to prevent man-in-the-middle attackers from accessing users' session details, and to prevent end users from examining the content in the JWT.

Perform the following steps to encrypt the JWT:

To Configure JWT Encryption

1. Navigate to Configure > Global Services > Session > Client-based Sessions.
2. From the Encryption Algorithm drop-down list, select one of the following algorithms:
   - **NONE**. No encryption algorithm is selected.
   - **RSA**. Session content is encrypted with AES using a unique key. The key is then encrypted with an RSA public key and appended to the JWT.
   - **AES KeyWrapping**. Session content is encrypted with AES using a unique key and is then wrapped using AES key wrap and the master key. This algorithm provides additional security,

compared to RSA, at the cost of 128 or 256 bits (or 32 bytes) depending on the size of the master key. It also provides authenticated encryption, which removes the need for a separate signature and decreases the byte size of the JWT.

- **Direct AES Encryption**. Session content is encrypted with direct AES encryption with a symmetric key. This method provides authenticated encryption, which removes the need for a separate signature and decreases the byte size of the JWT.

3. If you selected `RSA` in the previous step, you can select one of three padding options using the advanced property `org.forgerock.openam.session.stateless.rsa.padding`:

   - **RSA1_5**. RSA with PKCS#1 v1.5 padding.
   - **RSA-OAEP**. RSA with OAEP and SHA-1.
   - **RSA-OAEP-256**. RSA with OAEP padding and SHA-256.

   a. In the AM console, select Configure > Server Defaults > Advanced.

   b. In the Add a Value field, enter: `org.forgerock.openam.session.stateless.rsa.padding`.

   c. In the corresponding Add a Value field, enter one of the padding options. For example, `RSA-OAEP`. The default is `RSA-OAEP-256`.

   d. Click the plus sign ("+"), and then click Save Changes.

4. For the underlying content encryption method, select one of the following encryption methods supported in AM:

   - **A128CBC-HS256**. AES 128-bit in CBC mode with HMAC-SHA-256-128 hash (HS256 truncated to 128 bits)
   - **A192CBC-HS384**. AES 192-bit in CBC mode with HMAC-SHA-384-192 hash (HS384 truncated to 192 bits)
   - **A256CBC-HS512**. AES 256-bit in CBC mode with HMAC-SHA-512-256 hash (HS512 truncated to 256 bits)
   - **A128GCM**. AES 128-bit in GCM mode
   - **A192GCM**. AES 192-bit in GCM mode
   - **A256GCM**. AES 256-bit in GCM mode

   a. In the AM console, select Configure > Server Defaults > Advanced.

   b. In the Add a Value field, enter: `org.forgerock.openam.session.stateless.encryption.method`.

   c. In the corresponding Add a Value field, enter one of the padding options. For example, `A128CBC-HS512`. The default is `A128CBC-HS256`.

   d. Click the plus sign ("+"), and then click Save Changes.

5. In the Encryption RSA Certificate Alias field, enter an alias value to use for encrypting the JWT signature.

AM retrieves the certificate from the keystore specified by the `com.sun.identity.saml.xmlsig.keystore` property.

6. If you selected AES KeyWrapping or Direct AES Encryption, enter the key in the Encryption Symmetric AES Key field.

   This should be a base64-encoded random key. For direct encryption with AES-GCM or for AES-KeyWrap with any content encryption method, this should be 128, 192 or 256 bits.

   For direct encryption with AES-CBC-HMAC it should be double those sizes (one half for the AES key, the other half for the HMAC key). AES key sizes greater than 128 bits require installation of the JCE Unlimited Strength policy files in your JRE.

7. Save your changes.

8. Ensure that the JWT signature configuration is identical on every AM server in your AM site.

9. (Optional) To compress the session state, select `Deflate Compression` from the Compression Algorithm drop-down list.

   > **Warning**
   >
   > When set to `Deflate compression`, this option may lead to possible vulnerability with session state information leakage. Because the session token compression depends on the data in the session, an attacker can vary one part of the session (for example, the username or some other property) and then deduce some secret parts of the session state by examining how the session compresses. You should evaluate this threat depending on your use cases before enabling compression and encryption together.

   By default, AM rejects compressed session JWTs that expand to a size larger than 32 KiB (32768 bytes). For more information, see "Controlling the Maximum Size of Compressed JWTs" in the *Installation Guide*.

For detailed information about Session Service configuration attributes, see the entries for "Session" in the *Reference*.

Configuring Elliptic Curve Digital Signature Algorithms

AM supports Elliptic Curve Digital Signature Algorithms (ECDSA) as an alternative to RSA cryptography (RS256) or HMAC with SHA (HS256, HS384, HS512) signatures (see the JSON Web Algorithms specification, RFC 7518). The elliptic curve algorithms provide smaller key lengths for the same level of security that RSA provides (256-bit elliptic curve key vs 2048-bits RSA). The smaller key lengths result in faster signature and key generation times, and faster data transmission over TLS. One disadvantage for ECDSA is that signature verification can be significantly slower on the JVM.

AM supports the following elliptic curve signature algorithms:

- **ES256**. Elliptic Curve Digital Signature Algorithm (ECDSA) using SHA-256 hashes and the NIST standard P-256 elliptic curve. For more information on the NIST curves, see FIPS PUB 186-4: Digital Signature Standard (DSS), 2013-07-19.

- **ES384**. ECDSA using SHA-384 hashes and NIST standard P-384 curve.
- **ES512**. ECDSA using SHA-512 hashes and NIST standard P-521 curve.

To Configure Elliptic Curve Digital Signature Algorithms

1. Generate the public and private keys to use with the ECDSA algorithms using the standard curves parameters. You can use **keytool** to generate these key pairs. The following examples use a JCEKS keystore to store the keys:

   a. To generate an ES256-compatible keypair (picks the P-256 NIST curve):

   ```
   $ keytool -genkeypair -keystore mykeystore.jceks -alias ecdsa-test-cert -storepass xxx \
    -keypass yyy -dname 'CN=...' -storetype JCEKS -keyalg ec -keysize 256 \
    -validity 365
   ```

   b. To generate an ES384-compatible keypair (picks the P-384 NIST curve):

   ```
   $ keytool -genkeypair -keystore mykeystore.jceks -alias ecdsa-test-cert -storepass xxx \
    -keypass yyy -dname 'CN=...' -storetype JCEKS  -keyalg ec -keysize 384 \
    -validity 365
   ```

   c. To generate an ES512-compatible keypair (picks the P-521 NIST curve):

   ```
   $ keytool -genkeypair -keystore mykeystore.jceks -alias ecdsa-test-cert -storepass xxx \
    -keypass yyy -dname 'CN=...' -storetype JCEKS  -keyalg ec -keysize 521 \
    -validity 365
   ```

   > **Note**
   >
   > For ES512, the `-keysize` is `521`, not `512`.

2. Configure the ECDSA on AM:

   a. On the AM console, navigate to Configure > Global Services > Session. Click the Client-based Session tab.

   b. From the Signing Algorithm Type drop-down list, select the ECDSA algorithm that matches the alias in your keystore. For example, select `ES256` if you generated a ES256-compatible keypair.

   c. In the Signing RSA/ECDSA Certificate Alias field, enter the certificate alias that points to the ECDSA keypair.

3. Save your changes.

Configure Client-Based Session Security for Agents

To ensure that the client-based session cookie size does not surpass the browser supported size, Web Agents and Java Agents do not support both signing and encrypting the session cookie. To configure agents with client-based sessions, implement one of the following configurations:

- Configure signing and compression:

  1. Enable HS256 signing for the client-based session cookie. For more information, see "To Configure the JWT Signature".

  2. Enable compression by navigating to Configure > Global Services > Session > Client-based Sessions and selecting `Deflate Compression` from the Compression Algorithm drop-down.

- Configure encryption and compression:

  1. Set the `org.forgerock.openam.session.stateless.signing.allownone` advanced server property to `true` on every AM instance in the site.

  2. Disable signing for the client-based session cookie by navigating to Configure > Global Services > Session > Client-based Sessions and selecting `NONE` from the Signing Algorithm Type drop-down.

  3. Enable Direct AES Encryption. For more information, see "To Configure JWT Encryption".

  4. Enable compression by navigating to Configure > Global Services > Session > Client-based Sessions and selecting `Deflate Compression` from the Compression Algorithm drop-down.

Failure to set up client-based sessions correctly may cause unexpected errors when accessing a protected resource, such as blank pages and redirection loops.

Client-based sessions do not support restricted tokens. Therefore, Web Agents and Java Agents configured in a realm configured for client-based sessions are not protected against cookie hijacking. ForgeRock recommends using web or Java agents with CTS-based sessions.

Configuring Session Blacklisting

Session blacklisting ensures that users who have logged out of client-based sessions cannot achieve single sign-on without reauthenticating to AM. Session blacklisting does not apply to authentication sessions.

Perform the following steps to configure session blacklisting:

To Configure Session Blacklisting

1. Make sure that you deployed the Core Token Service during AM installation. The session blacklist is stored in the Core Token Service's token store.

2. Navigate to Configure > Global Services, click Session, and then locate the Client-based Sessions section.

3. Select the Enable Session Blacklisting option to enable session blacklisting for client-based sessions. When you configure one or more AM realms for client-based sessions, you should enable session blacklisting in order to track session logouts across multiple AM servers.

   Changing the value of this property takes effect immediately.

4. Configure the Session Blacklist Cache Size property.

   AM maintains a cache of logged out client-based sessions. The cache size should be around the number of logouts expected in the maximum session time. Change the default value of 10,000 when the expected number of logouts during the maximum session time is an order of magnitude greater than 10,000. An underconfigured session blacklist cache causes AM to read blacklist entries from the Core Token Service store instead of obtaining them from cache, which results in a small performance degradation.

   Changing the value of this property takes effect immediately.

5. Configure the Blacklist Poll Interval property.

   AM polls the Core Token Service for changes to logged out sessions if session blacklisting is enabled. By default, the polling interval is 60 seconds. The longer the polling interval, the more time a malicious user has to connect to other AM servers in a cluster and make use of a stolen session cookie. Shortening the polling interval improves the security for logged out sessions, but might incur a minimal decrease in overall AM performance due to increased network activity.

   Changing the value of this property does not take effect until you restart AM.

6. Configure the Blacklist Purge Delay property.

   When session blacklisting is enabled, AM tracks each logged out session for the maximum session time plus the blacklist purge delay. For example, if a session has a maximum time of 120 minutes and the blacklist purge delay is one minute, then AM tracks the session for 121 minutes. Increase the blacklist purge delay if you expect system clock skews in a cluster of AM servers to be greater than one minute. There is no need to increase the blacklist purge delay for servers running a clock synchronization protocol, such as Network Time Protocol.

   Changing the value of this property does not take effect until you restart AM.

7. Click Save Changes.

   Important

   Enabling or disabling the session blacklist, or changing the cache size takes effect immediately.

> Changes to any other session blacklist properties **do not** take effect until you restart AM.

8. Restart AM or the container where it runs.

   Any changes to the session blacklist properties only take effect after AM restarts.

For detailed information about Session Service configuration attributes, see the entries for "Session" in the *Reference*.

Limitations When Using Client-Based Sessions

The following AM features are not supported in realms that use client-based sessions:

- **Session quotas**. See "Implementing Session Quotas".
- **Session idle timeout**. See "Session Termination".
- **Cross-domain single sign-on with restricted tokens (Web Agents and Java Agents)**. See "Protecting Against Cookie Hijacking".
- **Both session signing and encryption (Web Agents and Java Agents)**. See "Configure Client-Based Session Security for Agents".
- **Uncompressed sessions (Web Agents and Java Agents)**. See "Configure Client-Based Session Security for Agents".
- **SAML v2.0 single logout using the SOAP binding**. See "Session State Considerations" in the *SAML v2.0 Guide*.
- **SAML 1.x single sign-on.** See the SAML v1.x Guide.
- **SNMP session monitoring.** See "SNMP Monitoring for Sessions" in the *Setup and Maintenance Guide*.
- **Session management by using the AM console.** See "Managing Sessions" in the *Setup and Maintenance Guide*.
- **Session notification.** See "Session" in the *Reference*.

Implementing In-Memory Authentication Sessions

Authentication chains always store authentication sessions in AM's memory. Perform the steps in the following procedure only for realms that configure authentication trees:

To Configure In-Memory Authentication Sessions

1. Ensure you have configured AM for sticky load-balancing. For more information, see "Configuring Load Balancing for a Site" in the *Installation Guide*.
2. Log in to the AM console as an administrative user, for example, `amAdmin`.
3. Navigate to Realms > *Realm Name* > Authentication > Settings > Trees.
4. From the Authentication session state management scheme drop-down list, select `In-Memory`.
5. In the Max duration (minutes) field, enter the maximum life of the authentication session in minutes.
6. Save your changes.
7. Navigate to Configure > Authentication > Core > Security.
8. In the Organization Authentication Signing Secret field, enter a base64-encoded HMAC secret that AM uses to sign the JWT that is passed back and forth between the client and AM during the authentication process. The secret must be, at least, 128-bits in length.
9. Save your changes.

Implementing Authentication Session Whitelisting

Enable authentication session whitelisting to protect authentication sessions from replay attacks.

When authentication session whitelisting is enabled, AM generates a key-value pair for each authentication session and stores it for the length of the authentication flow in the following ways:

- For client-based authentication sessions, AM stores the key-value pair in the CTS token store.
- For CTS-based authentication sessions, AM creates the key-value pair as a session property in the authentication session.
- For in-memory sessions, AM creates the key-value pair as a session property in the authentication session.

Each time the authentication flow reaches an authentication node, AM modifies the value of the stored key-value pair and sends it to the user or client that it is authenticating. The next request to AM to continue the authentication flow must contain the key-value pair and must match the value expected by AM.

If the authenticating user or client cannot provide the key-value pair with the values AM expects, AM would not continue the authentication flow, therefore protecting the authentication flow against malicious users wanting to rewind the authentication flow to a previous node.

Perform the following steps to configure authentication session whitelisting:

To Configure Authentication Session Whitelisting

1. Navigate to Realms > *Realm Name* > Authentication > Settings > Trees.
2. Select Enable whitelisting.
3. Save your changes.

Chapter 7
Implementing Single Sign-On

Single sign-on (SSO) allows a user or an entity to access multiple independent services from a single login session. AM supports SSO on the domain level and across multiple domains using cross-domain single sign-on (CDSSO).

About HTTP Cookies

To understand how SSO works, you need to understand some key elements of the HTTP cookie, as described in RFC 6525, HTTP State Management Mechanism .

Within an HTTP cookie, you can store a single custom *`name=value`* pair, such as *`sessionid=value`*. Other custom names within a cookie are as follows:

Domain

Normally set to the full URL that was used to access the configurator. To work with multiple subdomains, the `Domain` should be set to a URL like `Domain=server.example.net`. This is also known as the cookie domain.

Path

The directory in the URL to which the cookie applies. If the `Path` `=/openam`, the cookie applies to the `/openam` subdirectory of the URL, and lower level directories, including `openam/XUI`.

Secure

If the `Secure` name is included, the cookie can be transferred only over HTTPS. When a request is made over HTTP, the cookie is not made available to the application.

HttpOnly

When the `HttpOnly` flag is included, that cookie will not be accessible through JavaScript. According to RFC 6265, the noted flag "instructs the user agent to omit the cookie when providing access to cookies via 'non-HTTP' APIs (for example, a web browser API that exposes cookies to scripts)."

For more information, see "Configuring HttpOnly".

Expires

The lifetime of a cookie can be limited, with an `Expires` name configured with a time, based on UTC (GMT).

> **Warning**
>
> Do not take a shortcut with a top-level domain. Web browser clients today are designed to ignore cookies set to top-level domains including `com`, `net`, and `co.uk`. In addition, a cookie with a value like `Domain= app1.example.net` will not work for similar subdomains, such as `app2.example.net`.

Configuring HttpOnly

AM supports an `HttpOnly` flag, which is affixed to the `Set-cookie` HTTP response header transmitted from the server to the browser. The `HttpOnly` flag mitigates against cross-site scripting (XSS) vulnerabilities that can be exploited through JavaScript or other scripting languages.

When the `HttpOnly` flag is enabled:

- AM sets the token as HttpOnly. For example, AM returns a `Set-Cookie` header upon successful authentication. When HttpOnly is enabled, the header will include an `HttpOnly` flag with the original token in the payload of the `Set-Cookie` header as shown in the following example:

  ```
  Set-Cookie: iPlanetDirectoryPro='AQIC5..*; Domain=example.com; Path=/; HttpOnly
  ...
  ```

- When an invalid token is detected during the authentication process, the token is ignored and authentication continues. An additional `Set-Cookie` header is set to remove the invalid token from the client.

- Upon logout, the session cookie on the client is cleared by the `Set-Cookie` header in the response. For example:

  ```
  Set-Cookie: iPlanetDirectoryPro=""; Expires=Thu, 01 Jan 1970 00:00:10 GMT; Path=/; Domain=example.com;
   HttpOnly
  Set-Cookie: amlbcookie=LOGOUT; Expires=Thu, 01 Jan 1970 00:00:10 GMT; Path=/; Domain=example.com
  ...
  ```

- The User self-service auto login feature during the user registration process returns a `Set-Cookie` header in the response.

- Session upgrade automatically occurs upon the current SSO token when the authentication flow is called again and the token was previously passed in.

- When a client makes a call to the `/json/authenticate` endpoint appending a valid SSO token, AM returns the `tokenId` field **empty** when `HttpOnly` cookies are enabled. For example:

  ```
  {
      "tokenId":"",
      "successUrl":"/openam/console",
      "realm":"/"
  }
  ```

Perform the following steps to configure the `HttpOnly` flag:

To Configure the HttpOnly Flag

1. Log into the AM console as an administrator.
2. In the AM console, navigate to Configure > Server Defaults > Advanced.
3. Set the value of the `com.sun.identity.cookie.httponly` property to `true`.
4. Save your changes, and restart the AM instances.

Implementing Cross-Domain Single Sign-On

CDSSO provides SSO capabilities for AM servers and web or Java agents within a single domain or across domains in the same organization. For general information about how CDSSO works in AM, see "About Single Sign-On".

CDSSO is the only mode of operation of Web Agents and Java Agents and, therefore, no additional configuration is required to make it work. You must, however, protect the session cookie against hijacking.

Protecting Against Cookie Hijacking

A malicious user who steals a CDSSO cookie can potentially use it to access any realms that session has logged into, which may span multiple domains. For example, a token stolen from `myapp.example.com` could be used to access `payroll.internal.com` or any other protected domain in the same realm. Cookie hijacking protection restricts cookies to the fully-qualified domain name (FQDN) of the host where they are issued, such as `openam-server.example.com` and `server-with-agent.example.com`, using CDSSO to handle authentication and authorization.

For CDSSO with cookie hijacking protection, when a client successfully authenticates, AM issues the master SSO token cookie for its FQDN. AM issues *restricted token* cookies for the other FQDNs where the web or Java agents reside. The client ends up with cookies having different session identifiers for different FQDNs, and the AM server stores the correlation between the master SSO token and restricted tokens, such that the client only has one master session internally in AM.

To protect against cookie hijacking, you restrict the AM server domain to the server where AM runs. This sets the domain of the SSO token cookie to the host running the AM server that issued the token. You also enable use of a unique SSO token cookie. For your Java agents, you enable use of the unique SSO token cookie in the agent configuration.

> **Important**
>
> Client-based sessions do not support restricted tokens. Therefore, Web Agents and Java Agents configured in a realm configured for client-based sessions are not protected against cookie hijacking. ForgeRock recommends using web or Java agents with CTS-based sessions.

To Protect Against Cookie Hijacking

1. In the AM console, navigate to Configure > Global Services > Platform.

   a. Remove all domains from the Cookies Domains list.

   b. Save your work.

2. Navigate to Configure > Server Defaults > Advanced.

3. Set the `com.sun.identity.enableUniqueSSOTokenCookie` advanced property to `true`.

4. Save your work.

5. Restart AM or the container in which it runs for the configuration changes to take effect.

Troubleshooting SSO

In general, problems with single sign-on relate to some sort of mismatch of domain names. For example, a cookie that is configured on a third-level domain, such as `sso.example.net` will not work with an application on a similar domain, such as `app.example.net`. The following list describes scenarios that may lead to similar problems:

- When a cookie domain does not match a domain for the protected application.

  Assume the application is configured on a domain named `example.org`. That application will not receive an SSO token configured on the `example.net` domain.

- When a third-level domain is used for the SSO token.

  If an SSO token is configured on `sso.example.net`, an application on `app.example.net` does not receive the corresponding session token. In this case, the solution is to configure the SSO token on `example.net`.

- When the agent is configured for secure cookies (the `com.sun.identity.agents.config.cookie.secure` property is enabled) but the application makes HTTP requests.

  When enabled, agents mark cookies as secure, sending them only through a secure channel. Ensure your application communicates to the agent using HTTPS requests.

- When the path listed in the cookie does not match the path for the application.

Perhaps the cookie is configured with a `/helloworld` path; that will not match an application that might be configured with a `/hellomars` path. In that case, the application will not receive the cookie.

- When an inappropriate name is used for the cookie domain

  As noted earlier, client browsers are configured to ignore first-level domains, such as `com` and `net` as well as functional equivalents, such as `co.uk` and `co.jp`.

- When working with different browsers

  The *`name`*= *`value`* pairs described earlier may not apply to all browsers. The requirements for an HTTP cookie sent to an IE browser may differ from the requirements for other standard browsers, such as Firefox and Chrome. Based on anecdotal reports, IE does not recognize domain names that start with a number. In addition, IE reportedly refuses cookies that include the underscore (_) character in the FQDN.

- When a client-based session cookie exceeds the maximum size permitted by the browser

  As described in "Session Cookies", the default size of the `iPlanetDirectoryPro` cookie is approximately 2,000 bytes. When you customize AM sessions by adding attributes, the cookie size grows. Browsers allow cookie sizes between 4,000 and 5,200 bytes, depending on the browser. AM single sign-on does not support a cookie size that exceeds the maximum cookie size allowed by the browser.

Chapter 8
Using Authentication

This chapter covers how to authenticate and log out.

You can authenticate:

- From a browser, using the *extended user interface (XUI)*. See "Authenticating Using the XUI".
- By using the REST API. See "Authenticating by Using the REST API".

Tip

By default, AM redirects users to the console after successful authentication. No failure URL is defined by default. To configure success and failure URLs for a realm, see "Configuring Success and Failure Redirection URLs".

Authenticating From a Browser

You can use the XUI to authenticate from a browser.

Authenticating Using the XUI

When using the XUI, the base URL to authenticate to points to `/XUI/#login` under the deployment URL, such as `https://openam.example.com:8443/openam/XUI/#login`.

The base URL to log out is similar, for example, `https://openam.example.com:8443/openam/XUI/#logout/`.

Specifying the Realm in the Login URL

When making a request to the XUI, specify the realm or realm alias as the value of a `realm` parameter in the query string, or the DNS alias in the domain component of the URL. If you do not use a realm alias, then you must specify the entire hierarchy of the realm, starting at the top-level realm. For example `https://openam.example.com:8443/openam/XUI/?realm=/customers/europe#login/`.

The following table demonstrates additional examples:

Options for Specifying the Realm in XUI Login URLs

Description	Example URL
Full path of the realm as a parameter of `XUI`	`https://openam.example.com:8443/openam/XUI/?realm=/customers/europe#login`
Realm alias of the realm as a parameter of `XUI`	`https://openam.example.com:8443/openam/XUI/?realm=myrealm#login`
DNS Alias of the realm as the fully-qualified host name in the URL	`http://myRealm.example.com:8080/openam/XUI/#login`

The DNS alias is overridden by any use of either the full path or a realm alias as a query string parameter.

Example XUI Login URLs

Use any of the options listed in "Authentication Parameters" as URL parameters. Note that URL parameters must appear *before* any occurrences of the pound or hash character (**#**). The following are example URLs with parameters:

Example XUI Login URLs

Description	Example URL
Log in to the top level realm, requesting that AM display the user interface in German.	`https://openam.example.com:8443/openam/XUI/?realm=/&locale=de#login`
Log in to the `myRealm` subrealm whose parent is the top-level realm, requesting that AM display the user interface in German.	`https://openam.example.com:8443/openam/XUI/?realm=/myRealm&locale=de#login`
Log in to the `myRealm` subrealm whose parent is the top-level realm using the `HOTPChain` authentication chain, requesting that AM display the user interface in German.	`https://openam.example.com:8443/openam/XUI/?realm=/myRealm&locale=de&service=HOTPChain#login`

Authentication Parameters

AM accepts the following parameters in the query string. With the exception of `IDToken` parameters, use no more than one occurrence of each.

arg=newsession

Request that AM end the user's current session and start a new session.

authlevel

Request that AM authenticate the user using a module with at least the specified authentication level that you have configured.

As this parameter determines authentication module selection, do not use it with `module`, `service`, or `user`.

ForceAuth

If `ForceAuth=true`, request that AM force the user to authenticate even if they already has a valid session. On successful authentication, AM does one of the following:

- (Authentication trees only) AM issues new session tokens to users on reauthentication, even if the current session already meets the security requirements.
- (Authentication chains only) AM does not issue new session tokens on reauthentication, regardless of the security level they are authenticating to. Instead, it updates the session token with the new authentication information, if required.

> **Important**
>
> For authentication chains, the forceAuth.enabled advanced server property controls whether the value of the `ForceAuth` parameter is respected.

goto

On successful authentication, or successful logout, request that AM redirect the user to the specified location. Values must be URL-encoded. See "Configuring Success and Failure Redirection URLs" for more information.

gotoOnFail

On authentication failure, request that AM redirect the user to the specified location. Values must be URL-encoded. See "Configuring Success and Failure Redirection URLs" for more information.

IDToken1, IDToken2, ..., IDTokenN

Pass the specified credentials as `IDToken` parameters in the URL. The `IDToken` credentials map to the fields in the login page for the authentication module, such as `IDToken1` as user ID and `IDToken2` as password for basic user name, password authentication. The order depends on the callbacks in login page for the module; `IDTokenN` represents the Nth callback of the login page.

locale

Request that AM display the user interface in the specified, supported locale. Locale can also be set in the user's profile, in the HTTP header from her browser, configured in AM, and so on.

module

Request that AM use the authentication module instance as configured for the realm where the user is authenticating.

As this parameter determines authentication module selection, do not use it with `authlevel`, `service`, or `user`.

realm

Request that AM authenticate the user to the specified realm.

resource

Set this parameter to `true` to request resource-based authentication.

For resource-based authentication, also set the `resourceURL` parameter.

resourceURL

Set this parameter to the URL of the resource for resource-based authentication.

Resource-based authentication applies when an authorization policy has an environment setting of type Authentication by Module Chain or Authentication by Module Instance. When the specified resource URL matches a policy resource, AM finds the chain or module configured in the policy environment settings. AM then uses the specified chain or module to perform authentication.

For example, if you configure a policy with the resource `https://www.example.com:443/index.html` and the environment Authentication by Module Chain: DataStore, then the following login URL causes AM to use the DataStore chain to authenticate the user:

```
https://openam.example.com:8443/openam/XUI/?resource=true&resourceURL=https://www.example.com:443/
index.html&goto=https://www.example.com/
```

On successful authentication, AM redirects the user-agent to `https://www.example.com/`.

As shown in the example, when setting the `resourceURL` parameter, also set `resource=true`.

service

Request that AM authenticate the user with the specified authentication chain.

As this parameter determines authentication module selection, do not use it with `authlevel`, `module`, or `user`.

user

Request that the user, specified by their AM universal ID, authenticates according to the chain specified by the User Authentication Configuration property in their user profile. You can configure this property for a user under Realms > *Realm Name* > Identities > *UserName*.

In order for the User Authentication Configuration property to appear in user profiles, the `iplanet-am-user-service` object class must contain the `iplanet-am-user-auth-config` attribute in the identity repository schema. The default identity repository schemas provided with AM include this object class and attribute. See "Preparing Identity Repositories" in the *Installation Guide* for information about identity repository schema.

As this parameter determines authentication module selection, do not use it with authlevel, module, or service.

Authenticating by Using the REST API

For information about how to authenticate to AM using the REST API, see "Authentication and Logout using REST".

For information about how to use the session token returned from the REST API when authentication is successful, see "Using the Session Token After Authentication".

Sample Mobile Authentication Applications

Source code for sample mobile applications is available in sample repositories in the ForgeRock commons project. Get local clones of one or more of the following repositories so that you can try these sample applications on your system:

- AM access from iOS
- AM single sign-on from iOS
- AM authentication and logout using PhoneGap

Configuring Success and Failure Redirection URLs

AM determines the redirection URL based on authentication success or failure. During success, AM redirects the user to the URL specified in the goto parameter and, during failure, AM redirects the user to the URL specified in the gotoOnFail parameter.

AM provides a number of places where you can configure success or failure URLs:

Successful Authentication URL Precedence

Upon a successful authentication, AM determines the redirection URL in the following order:

1. The URL set in the authentication chain or authentication tree.
   - To specify a URL in an authentication chain, in the AM console, set the Successful Login URL parameter by navigating to *Realm Name* > Authentication > Chains > *chain* > Settings.
   - To specify a URL in an authentication tree, add a Success URL Node to the tree and specify the Success URL in the node properties.
2. The URL set in the goto login URL parameter. For example:

```
https://openam.example.com:8443/openam/XUI/?realm=/&goto=http%3A%2F%2Fwww.example.com#login
```

3. The URL set in the Success URL attribute in the user's profile.

   In the AM console, you can set the Success URL parameter by navigating to *Realm Name* > Identities > *identity*. In Success URL, enter a URL, and then click Save Changes.

   You can also specify the client type by entering `ClientType|URL` as the property value. If the client type is specified, it will have precedence over a regular URL in the user's profile.

4. The URL set in the Default Success Login URL attribute in the Top Level realm.

   You can set this property on the AM console by navigating to Configure > Authentication > Core Attributes > Post Authentication Processing.

   You can also specify the client type by entering `ClientType|URL` as the property value. If the client type is specified, it will have precedence over a Default Success Login URL in the Top Level realm.

Failed Authentication URL Precedence

Upon a failed authentication, AM determines the redirection URL in the following order:

1. The URL set in the authentication chain or authentication tree.

   - To specify a URL in an authentication chain, in the AM console, set the Failed Login URL parameter by navigating to *Realm Name* > Authentication > Chains > *chain* > Settings.

   - To specify a URL in an authentication tree, add a Failure URL Node to the tree and specify the Failure URL in the node properties.

2. The URL set in the `gotoOnFail` parameter. For example:

   ```
   https://openam.example.com:8443/openam/XUI/?realm=/&gotoOnFail=http%3A%2F%2Fwww.example.com#login
   ```

3. The URL set in the Failure URL attribute in the user's profile.

   In the AM console, you can set the Failure URL parameter by navigating to *Realm Name* > Identities > *identity*. Under Failure URL, enter a URL, and then click Save Changes.

   You can also specify the client type by entering `ClientType|URL` as the property value. If the client type is specified, it will have precedence over a regular URL in the user's profile.

4. The URL set in the Default Failure Login URL attribute in the Top Level realm.

   You can set this property on the AM console by navigating to Configure > Authentication > Core Attributes > Post Authentication Processing.

   You can also specify the client type by entering `ClientType|URL` as the property value. If the client type is specified, it will have precedence over a Default Failure Login URL in the Top Level realm.

URLs can be relative to AM's URL, or absolute.

By default, AM trusts all relative URLs and those absolute URLs that are in the same scheme, FQDN, and port as AM. This increases security against possible phishing attacks through open redirect.

To configure AM to trust other absolute URLs, add them to the Validation Service. If you do not, on login AM will redirect to the user profile or to the administrator console, and on logout to the default logout page in the UI instead.

+ *Do I Need to Add my URL to the Validation Service?*

Consider an example AM deployment configured in `https://openam.example.com:8443/openam`:

URL	Needs to be configured in the Validation Service?
`http://openam.example.com:8080/openam/XUI/#login`	Yes, the scheme and port are different.
`https://openam.example.com:443/openam/XUI/#login`	Yes, the port is different.
`/openam/XUI/#login`	No, the paths relative to the AM URL are trusted.
`https://mypage.example.com/app/logout.jsp`	Yes, the scheme, port, and FQDN are different.

To Configure the Validation Service

1. In the AM console, navigate to Realms > *Realm Name* > Services.

   Note that you can add an instance of the Validation Service on the Top Level Realm, too.

2. Click Add a Service.

3. From the Choose a service type drop-down list, select Validation Service.

4. In the Valid goto URL Resources field, enter one or more valid URL patterns to allow.

   For example, `http://app.example.com:80/*?*`

   For information on pattern matching and wildcard rules, see "Specifying Resource Patterns with Wildcards" in the *Authorization Guide*.

   + *General Examples of URL Pattern Matching*

     - If no port is specified, `http://www.example.com` canonicalizes to `http://www.example.com:80` and `https://www.example.com` canonicalizes to `http://www.example.com:443`.
     - A wildcard before "://" only matches up to "://"

For example, `http*://*.com/*` matches `http://www.example.com/hello/world` and `https://www.example.com/hello`.

- A wildcard between "://" and ":" matches up to ":"

  For example, `http://*:85` matches `http://www.example.com:85`.

- A wildcard between ":" and "/" only matches up to the first "/"

  For example, `http://www.*:*/` matches `http://www.example.com:80`. In another example, `http://www.example.com:*` matches `http://www.example.com:[any port]` and `http://www.example.com:[any port]/`, but nothing more.

- A wildcard after "/" matches anything, depending on whether it is single-level or a wildcard appropriately.

  For example, `https://www.example.com/*` matches `https://www.example.com:443/foo/bar/baz/me`

- If you do not use any wildcards, AM exactly matches the string, so `http://www.example.com` only matches `http://www.example.com`, but NOT `http://www.example.com/` (trailing slash).

  If you put the wildcard after the path, AM expects a path (even if it is blank), so `http://www.example.com/*` matches `http://www.example.com/` and `http://www.example.com/foo/bar/baz.html`, but NOT `http://www.example.com`.

- `http://www.example.com:*/` matches `http://www.example.com/`, which also canonicalizes to `http://www.example.com:80/`.

- `https://www.example.com:*/` matches `https://www.example.com/`, which also canonicalizes to `https://www.example.com:443/`.

5. Click Create to save your settings.

Tip

To validate a goto URL over REST, use the endpoint: `/json/users?_action=validateGoto`.

```
$ curl \
--request POST \
--header "Accept-API-Version: protocol=2.1,resource=3.0" \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5...ACMDE.*" \
--data '{"goto":"http://www.example.com/"}' \
https://openam.example.com:8443/openam/json/users?_action=validateGoto
{
    "successURL":"http://www.example.com/"
}
```

Chapter 9
Managing Sessions (REST)

AM provides REST APIs under /json/sessions for validating SSO tokens and getting information about active sessions.

Obtaining Information About Sessions

To get information about a session, send an HTTP POST to the /json/sessions/ endpoint, using the getSessionInfo action. This endpoint returns information about the session token provided in the iPlanetDirectoryPro header by default. To get information about a different session token, include it in the POST body as the value of the tokenId parameter.

The following example shows an administrative user passing their session token in the iPlanetDirectoryPro header, and the session token of the demo user as the tokenId parameter:

```
$ curl \
--request POST \
--header "iplanetDirectoryPro: AQIC4Dm...NTcy*" \
--header "Accept-API-Version: resource=4.0" \
--header "Content-Type: application/json" \
--data '{ "tokenId": "BXCCq...NX*1*" }' \
https://openam.example.com:8443/openam/json/realms/root/sessions/?_action=getSessionInfo
{
    "username": "demo",
    "universalId": "id=demo,ou=user,dc=openam,dc=forgerock,dc=org",
    "realm": "/",
    "latestAccessTime": "2020-02-21T14:31:18Z",
    "maxIdleExpirationTime": "2020-02-21T15:01:18Z",
    "maxSessionExpirationTime": "2020-02-21T16:29:56Z",
    "properties": {
        "AMCtxId": "aba7b4f3-16ff-4680-b06a-d7ba237d3730-91932"
    }
}
```

The getSessionInfo action does not refresh the session idle timeout. To obtain session information about a CTS-based session and also reset the idle timeout (you cannot reset the idle timeout of client-based sessions), use the getSessionInfoAndResetIdleTime endpoint, as follows:

```
$ curl \
--request POST \
--header "iplanetDirectoryPro: AQIC4Dm...NTcy*" \
--header "Accept-API-Version: resource=4.0, protocol=1.0" \
--header "Content-Type: application/json" \
--data '{ "tokenId": "BXCCq...NX*1*" }' \
https://openam.example.com:8443/openam/json/realms/root/sessions/?_action=getSessionInfoAndResetIdleTime
{
    "username": "demo",
    "universalId": "id=demo,ou=user,dc=openam,dc=forgerock,dc=org",
    "realm": "/",
    "latestAccessTime": "2020-02-21T14:32:49Z",
    "maxIdleExpirationTime": "2020-02-21T15:02:49Z",
    "maxSessionExpirationTime": "2020-02-21T16:29:56Z",
    "properties": {
        "AMCtxId": "aba7b4f3-16ff-4680-b06a-d7ba237d3730-91932"
    }
}
```

Note

In order to return the AMCtxId property in the session query response, as in this example, you must set AMCtxId in the Session Properties to return to session queries setting, under Realms > *Realm Name* > Services > Session Property Whitelist Service.

Validating Sessions

To check over REST whether a session token is valid, perform an HTTP POST to the /json/sessions/ endpoint using the validate action. Provide the session token in the POST data as the value of the tokenId parameter. You must also provide the session token of an administrative user in the iPlanetDirectoryPro header.

If you don't specify the tokenId parameter, the session in the iPlanetDirectoryPro header is validated instead.

The following example shows an administrative user, such as amAdmin, validating a session token for the demo user:

```
$ curl \
--request POST \
--header "Content-type: application/json" \
--header "iPlanetDirectoryPro: AQICS...NzEz*" \
--header "Accept-API-Version: resource=2.1, protocol=1.0" \
--data '{ "tokenId": "S9DSqBv...AlMxAAA.*" }' \
https://openam.example.com:8443/openam/json/realms/root/sessions?_action=validate
```

If the session token is valid, the user ID and its realm is returned, as shown below:

```
{
  "valid":true,
  "sessionUid":"209331b0-6d31-4740-8d5f-740286f6e69f-326295",
  "uid":"demo",
  "realm":"/"
}
```

By default, validating a session resets the session's idle time, which triggers a write operation to the Core Token Service token store. To avoid this, perform a call using the `validate&refresh=false` action.

Refreshing CTS-Based Sessions

To reset the idle time of a CTS-based session using REST, perform an HTTP POST to the `/json/sessions/` endpoint, using the `refresh` action. The endpoint will refresh the session token provided in the `iPlanetDirectoryPro` header by default. To refresh a different session token, include it in the POST body as the value of the `tokenId` query parameter.

The following example shows an administrative user passing their session token in the `iPlanetDirectoryPro` header, and the session token of the `demo` user as the `tokenId` parameter:

```
$ curl \
--request POST \
--header 'Content-Type: application/json' \
--header "iplanetDirectoryPro: AQIC5w...NTcy*" \
--header "Accept-API-Version: resource=3.1, protocol=1.0" \
--data '{ "tokenId": "BXCCq...NX*1*" }' \
https://openam.example.com:8443/openam/json/realms/root/sessions/?_action=refresh
{
  "uid": "demo",
  "realm": "/",
  "idletime": 17,
  "maxidletime": 30,
  "maxsessiontime": 120,
  "maxtime": 7106
}
```

On success, AM resets the idle time for the CTS-based session, and returns timeout details of the session.

Resetting a CTS-based session's idle time triggers a write operation to the Core Token Service token store. Therefore, to avoid the overhead of write operations to the token store, be careful to use the `refresh` action only if you want to reset a CTS-based session's idle time.

Because AM does not monitor idle time for client-based sessions, do not use the `tokenId` of a client-based session when refreshing a session's idle time.

Invalidating Sessions

To invalidate a session, perform an HTTP POST to the `/json/sessions/` endpoint using the `logout` action. The endpoint will invalidate the session token provided in the `iPlanetDirectoryPro` header:

```
$ curl \
--request POST \
--header "iplanetDirectoryPro: BXCCq...NX*1*" \
--header "Accept-API-Version: resource=3.1, protocol=1.0" \
https://openam.example.com:8443/openam/json/realms/root/sessions/?_action=logout
{
    "result": "Successfully logged out"
}
```

On success, AM invalidates the session and returns a success message.

If the token is not valid and cannot be invalidated an error message is returned, as follows:

```
{
  "result": "Token has expired"
}
```

To invalidate a different session token, include it in the POST body as the value of the `tokenId` parameter.

For example, the following command shows an administrative user passing their session token in the `iPlanetDirectoryPro` header, and the session token of the `demo` user as the `tokenId` parameter:

```
$ curl \
--request POST \
--header "iplanetDirectoryPro: AQIC5w...NTcy*" \
--header "Accept-API-Version: resource=3.1, protocol=1.0" \
--data '{ "tokenId": "BXCCq...NX*1*" }' \
"https://openam.example.com:8443/openam/json/realms/root/sessions/?_action=logout"
{
    "result": "Successfully logged out"
}
```

Getting and Setting Session Properties

AM lets you read and update properties on users' sessions using REST API calls.

Before you can perform operations on session properties using the REST API, you must first define the properties you want to set in the Session Property Whitelist Service configuration. For information on whitelisting session properties, see "Session Property Whitelist Service" in the *Reference*.

You can use REST API calls for the following purposes:

- To retrieve the names of the properties that you can read or update. This is the same set of properties configured in the Session Property Whitelist Service.
- To read property values.
- To update property values.

Session state affects the ability to set and delete properties as follows:

- You can set and delete properties on a CTS-based session at any time during the session's lifetime.
- You can only set and update properties on a client-based session during the authentication process, before the user receives the session token from AM. For example, you could set or delete properties on a client-based session from within a post-authentication plugin.

Differentiate the user who performs the operation on session properties from the session affected by the operation as follows:

- Specify the session token of the user performing the operation on session properties in the `iPlanetDirectoryPro` header.
- Specify the session token of the user whose session is to be read or modified as the `tokenId` parameter in the body of the REST API call.
- Omit the `tokenId` parameter from the body of the REST API call if the session of the user performing the operation is the same session that you want to read or modify.

The following examples assume that you configured a property named `LoginLocation` in the Session Property Whitelist Service configuration.

To retrieve the names of the properties you can get or set, and their values, perform an an HTTP POST to the sessions endpoint, `/json/sessions/`, using the `getSessionProperties` action, as shown in the following example:

```
$ curl \
--request POST \
--header "iplanetDirectoryPro: AQIC5w...NTcy*" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=3.1, protocol=1.0" \
--data '{ "tokenId": "BXCCq...NX*1*" }' \
https://openam.example.com:8443/openam/json/realms/root/sessions/?_action=getSessionProperties
{
    "LoginLocation": ""
}
```

To set the value of a session property, perform an HTTP POST to the sessions endpoint, `/json/sessions/`, using the `updateSessionProperties` action. If no `tokenId` parameter is present in the body of the REST API call, the session affected by the operation is the session specified in the `iPlanetDirectoryPro` header, as follows:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iplanetDirectoryPro: BXCCq...NX*1*" \
--header "Accept-API-Version: resource=3.1, protocol=1.0" \
--data '{"LoginLocation":"40.748440, -73.984559"}' \
https://openam.example.com:8443/openam/realms/root/json/sessions/?_action=updateSessionProperties
{
    "LoginLocation": "40.748440, -73.984559"
}
```

You can set multiple properties in a single REST API call by specifying a set of fields and their values in the JSON data. For example:

```
--data '{"property1":"value1", "property2":"value2"}'
```

To set the value of a session property on another user's session, specify the session token of the user performing the `updateSessionProperties` action in the `iPlanetDirectoryPro`, and specify the session token to be modified in the POST body as the value of the `tokenId` parameter:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iplanetDirectoryPro: AQIC5w...NTcy*" \
--header "Accept-API-Version: resource=3.1, protocol=1.0" \
--data '{"LoginLocation": "40.748440, -73.984559", "tokenId": "BXCCq...NX*1*"}' \
https://openam.example.com:8443/openam/json/realms/root/sessions/?_action=updateSessionProperties
{
    "LoginLocation": "40.748440, -73.984559"
}
```

If the user attempting to modify the session does not have sufficient access privileges, the preceding examples result in a 403 Forbidden error.

You cannot set properties internal to AM sessions. If you try to modify an internal property in a REST API call, a 403 Forbidden error is returned. For example:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iplanetDirectoryPro: AQIC5w...NTcy*" \
--header "Accept-API-Version: resource=3.1, protocol=1.0" \
--data '{"AuthLevel":"5", "tokenId": "BXCCq...NX*1*"}' \
https://openam.example.com:8443/openam/json/realms/root/sessions/?_action=updateSessionProperties
{
    "code": 403,
    "reason": "Forbidden",
    "message": "Forbidden"
}
```

Performing Session Upgrade

This section demonstrates how to set up AM to allow users to upgrade their sessions during policy evaluation. For general information information about session upgrade, see "Session Upgrade".

Session Upgrade Prerequisites

Before attempting to upgrade sessions, perform the following prerequisite tasks:

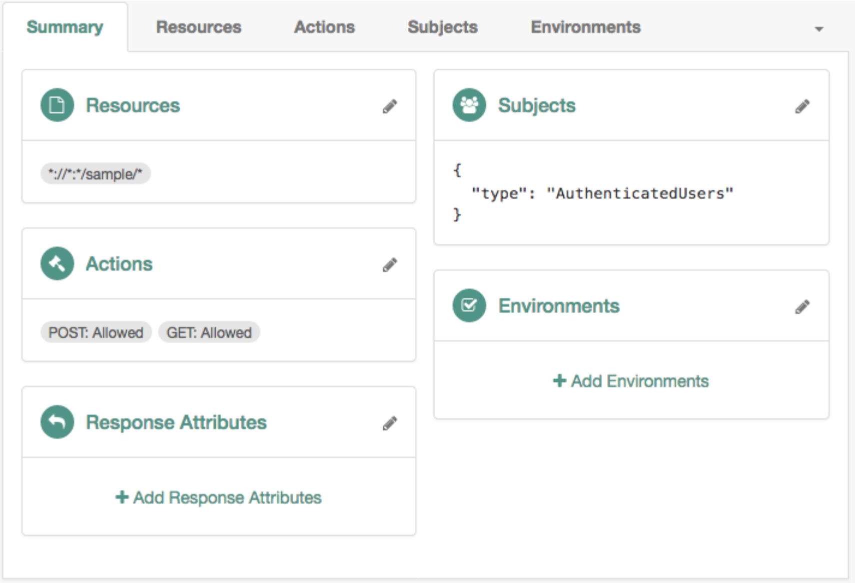
- Configure a policy enforcement point (PEP), for example, a web or Java agent, that enforces AM policies on a website or application.

AM web and Java agents handle session upgrade without additional configuration because the agents are built to handle AM's advices. If you build your own PEPs, however, you must take advices and session upgrade into consideration.

- For information about configuring web agents, see the *ForgeRock Web Policy Agent documentation*.
- For information, about configuring Java agents, see the *ForgeRock Java Agent documentation*.
- For information on how to handle advices on RESTful PEPs, see "Requesting Policy Decisions" in the *Authorization Guide*.

- Configure an authorization policy to protect a resource protected by the Java or web agent, or a RESTful PEP. The following example of authorization policy allows GET and POST access to the `*://*:*/sample/*` resource to any authenticated user:

Authorization Policy Example



```
{
  "type": "AuthenticatedUsers"
}
```

Once you have set up your environment, ensure that you are required to log in to access your protected resource.

Tip

To configure your environment quickly with a web agent to test session upgrade, see "Setting Up the Software" in the *Quick Start Guide*.

Configuring the Environment for Session Upgrade

To configure your environment to attempt a session upgrade, perform the steps in the following procedure:

To Configure the Environment for Session Upgrade

1. Configure an authentication tree or chain to validate users' credentials during session upgrade.

   Authentication trees and chains do not require additional configuration to perform session upgrade. However, because session upgrade is a mechanism which may be used to grant users access to sensitive information, you should consider configuring a strong authentication method such as multi-factor authentication. Also, you may want to consider how long-lived sessions in your environment are. For example, if users should only have access to the protected resource to perform an operation, such as check the balance of an account, you may want to consider implementing transactional authorization instead. For more information, see "*Implementing Transactional Authorization*" in the *Authorization Guide*.

   - For more information about configuring authentication trees, see "Configuring Authentication Trees".

   - For more information about configuring authentication chains, see "Configuring Authentication Chains and Modules".

2. Configure at least one of the following environment conditions in the authentication policy created as part of the prerequisites:

   - Authentication Level (greater than or equal to)

   - Authentication by Module Instance (Authentication modules only)

   - Authentication by Service

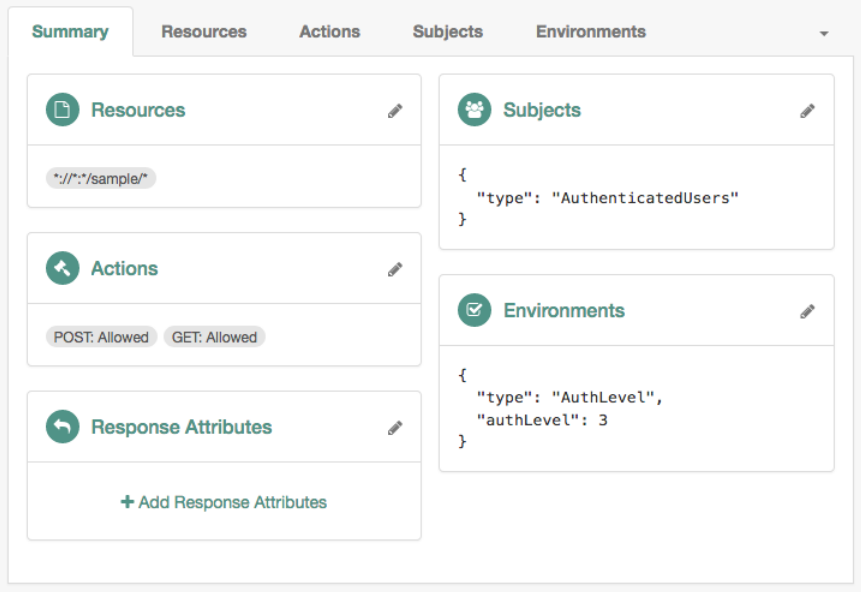
   Note

   The following examples feature simple policy conditions. For more information about configuring policies and environment conditions, see "Configuring Policies" in the *Authorization Guide*.

   Authentication Level (greater than or equal to) *Authentication modules only*

   Use this condition to present a list of authentication chains that provide a greater or equal authentication level to the one specified in the condition. The user selects their service of choice if multiple chains are able to meet the criteria of the condition. For example, the following policy requires a chain that provides authentication level 3 or greater:

Session Upgrade by Authentication Level (greater than or equal to)



```
{
  "type": "AuthenticatedUsers"
}
```

```
{
  "type": "AuthLevel",
  "authLevel": 3
}
```

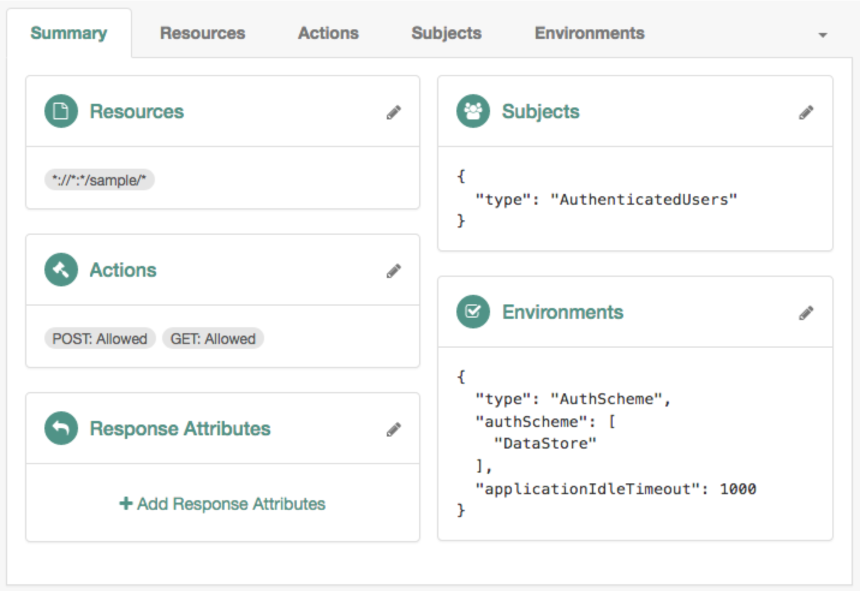
Tip

For more information about configuring the authentication level by authentication module, see "Auth Level Decision Node".

Authentication by Module Instance *Authentication modules only*

Use this condition to enforce that a user has gone through a specific authentication module. For example, the following policy requires the user to log in with the DataStore module:

Session Upgrade by Module Instance



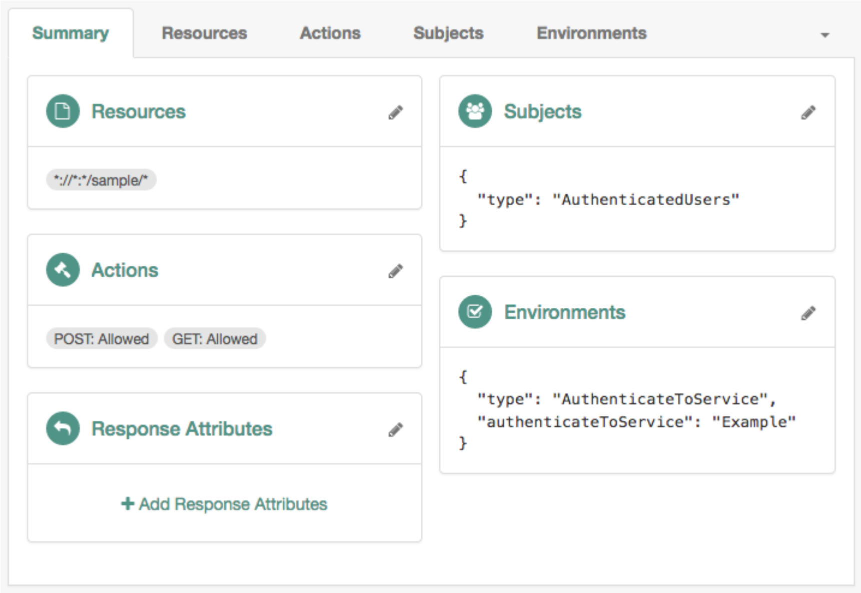
```
{
  "type": "AuthenticatedUsers"
}
```

```
{
  "type": "AuthScheme",
  "authScheme": [
    "DataStore"
  ],
  "applicationIdleTimeout": 1000
}
```

Authentication by Service

Use this condition to specify the chains or authentication trees to which the user needs to use to authenticate. For example, the following policy requires the user to log in with the `Example` tree:

Session Upgrade by Service



://:*/sample/*

```
{
  "type": "AuthenticatedUsers"
}
```

POST: Allowed GET: Allowed

```
{
  "type": "AuthenticateToService",
  "authenticateToService": "Example"
}
```

Note that the names of the authentication trees and chains are case-sensitive.

3. Test session upgrade:

   - To test session upgrade with a browser, see "Performing Session Upgrade Using a Browser".
   - To test session upgrade with REST, see "Performing Session Upgrade Using REST".

Performing Session Upgrade Using a Browser

The following procedure demonstrates how to upgrade a session when using a browser:

To Perform Session Upgrade Using a Browser

To upgrade a session using a browser, perform the following steps:

1. Ensure you have performed the tasks in "Session Upgrade Prerequisites" and "To Configure the Environment for Session Upgrade".

2. In a browser, navigate to your protected resource. For example, `http://www.example.com:9090/sample`.

   The agent redirects the browser to the AM login screen.

3. Authenticate to AM as the `demo` user.

   AM requires additional credentials to grant access to the resource. For example, if you set the policy environment condition to `Authentication by Service` and `Example`, you will be required to log in again as the `demo` user.

4. Authenticate as the `demo` user. Note that providing credentials for a different user will fail.

   You can now access the protected resource.

Performing Session Upgrade Using REST

The following procedure demonstrates the REST flow to upgrade a session:

To Perform Session Upgrade Using REST

To upgrade a session using REST, perform the following steps:

1. Ensure you have performed the tasks in "Session Upgrade Prerequisites" and "To Configure the Environment for Session Upgrade".

   Note

   This example uses composite advice with an authentication level condition, which only applies to authentication chains.

2. Log in with an administrative user that has permission to evaluate policies, such as `amAdmin`. For example:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: amadmin" \
--header "X-OpenAM-Password: password" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
{
    "tokenId":"AQIC5wM2...",
    "successUrl":"/openam/console",
    "realm":"/"
}
```

   Tip

   You can also assign privileges to a user to evaluate policies. For more information, see "To Allow a User to Evaluate Policies" in the *Authorization Guide*.

3. Log in with the user that should access the resources. For example, log in as the `demo` user:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: demo" \
--header "X-OpenAM-Password: changeit" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
{
    "tokenId":"AQIC5wM...TU3OQ*",
    "successUrl":"/openam/console",
    "realm":"/"
}
```

4. Request a policy decision from AM for a protected resource, in this case, `http://openam.example.com:9090/sample`. The `iPlanetDirectoryPro` header sets the SSO token for the administrative user, and the `subject` element of the payload sets the SSO token for the `demo` user:

```
$ curl --request POST \
 --header "Content-Type: application/json" \
 --header "iPlanetDirectoryPro: AQIC5wM2..." \
 --header "Accept-API-Version:protocol=1.0,resource=2.1" \
 --data '{
 "resources": [
     "http://www.example.com:9090/sample"
 ],
 "application": "iPlanetAMWebAgentService",
 "subject": { "ssoToken": "AQIC5wM...TU3OQ*"}
}' \
"https://openam.example.com:8443/openam/json/policies?_action=evaluate"
[
   {
      "resource":"http://www.example.com:9090/sample",
      "actions":{

      },
      "attributes":{

      },
      "advices":{
         "AuthLevelConditionAdvice":[
            "3"
         ]
      },
      "ttl":9223372036854775807
   }
]
```

AM returns with advice, which means the user must present additional credentials to access that resource.

For more information about requesting policy decision, see "Requesting Policy Decisions" in the *Authorization Guide*.

5. Format the advice as XML, without spaces or line breaks. The following example is spaced and tabulated for readability purposes only:

```
<Advices>
    <AttributeValuePair>
       <Attribute name="AuthLevelConditionAdvice"/>
       <Value>3</Value>
    </AttributeValuePair>
</Advices>
```

> **Note**
>
> The example shows the XML render of a single advice. Depending on the conditions configured in the policy, the advice may contain several lines. For more information about advices, see "Policy Decision Advice" in the *Authorization Guide*.

6. URL-encode the XML advice. For example: `%3CAdvices%3E%3CAttributeValuePair%3E%3CAttribute%20name%3D%22AuthLevelConditionAdvice%22%2F%3E%3CValue%3E3%3C%2FValue%3E%3C%2FAttributeValuePair%3E%3C%2FAdvices%3E`.

   Ensure there are no spaces between tags when URL-encoding the advice.

7. Call AM's `authenticate` endpoint to request information about the advice. Use the following details:

   - Add the following URL parameters:
     - `authIndexType=composite_advice`
     - `authIndexValue=`*`URL-encoded_Advice`*
   - Set the `iPlanetDirectoryPro` cookie as the SSO token for the `demo` user.

   For example:

```
$ curl --request POST \
--header "Content-Type: application/json" \
--cookie "iPlanetDirectoryPro=AQIC5wM...TU3OQ*" \
--header "Accept-API-Version: protocol=1.0,resource=2.1" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate?
authIndexType=composite_advice&amp;authIndexValue=%3CAdvices%3E%3CAttributeValuePair%3E...'
{
   "authId":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhdXRoSW5kZ...",
   "template":"",
   "stage":"DataStore1",
   "header":"Sign in",
   "callbacks":[
      {
         "type":"NameCallback",
         "output":[
            {
               "name":"prompt",
               "value":"User Name:"
            }
         ],
         "input":[
            {
               "name":"IDToken1",
               "value":""
            }
         ]
      },
      {
         "type":"PasswordCallback",
         "output":[
            {
               "name":"prompt",
               "value":"Password:"
            }
         ],
         "input":[
            {
               "name":"IDToken2",
               "value":""
            }
         ]
      }
   ]
}
```

AM returns information about how the user can authenticate in a callback; in this case, providing a username and password. For a list of possible callbacks, and more information about the `/json/authenticate` endpoint, see "Authentication and Logout using REST".

8. Call AM's `authenticate` endpoint to provide the required callback information. Use the following details:

   - Add the following URL query parameters:

     - `authIndexType=composite_advice`

     - `authIndexValue=`*`URL-encoded_Advice`*

   - Set the `iPlanetDirectoryPro` cookie as the SSO token for the `demo` user.

   - Send as data the complete payload AM returned in the previous step, ensuring you provide the requested callback information.

   In this example, provide the username and password for the `demo` user in the `input` objects, as follows:

```
$ curl --request POST \
     --header 'Content-Type: application/json' \
     --header "Accept-API-Version: protocol=1.0,resource=2.1" \
     --cookie "iPlanetDirectoryPro=AQIC5wM...TU3OQ*" \
     --data '{
                "authId":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhdXRoSW5kZ...",
                "template":"",
                "stage":"DataStore1",
                "header":"Sign in",
                "callbacks":[
                   {
                      "type":"NameCallback",
                      "output":[
                         {
                            "name":"prompt",
                            "value":"User Name:"
                         }
                      ],
                      "input":[
                         {
                            "name":"IDToken1",
                            "value":"demo"
                         }
                      ]
                   },
                   {
                      "type":"PasswordCallback",
                      "output":[
                         {
                            "name":"prompt",
                            "value":"Password:"
                         }
                      ],
                      "input":[
                         {
                            "name":"IDToken2",
                            "value":"changeit"
                         }
                      ]
                   }
                ]
             }
           }' \
     'https://openam.example.com:8443/openam/json/realms/root/authenticate?
authIndexType=composite_advice&amp;authIndexValue=%3CAdvices%3E%3CAttributeValuePair%3E...'
{
   "tokenId":"wpU01SaTq4X2x...NDVFMAAlMxAAA.*",
   "successUrl":"/openam/console",
   "realm":"/"
}
```

Note that AM returns a new SSO token for the `demo` user.

9. Request a new policy decision from AM for the protected resource. The `iPlanetDirectoryPro` header sets the SSO token for the administrative user, and the subject element of the payload sets the new SSO token for the demo user:

```
$ curl --request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5wM2..." \
--header "Accept-API-Version:protocol=1.0,resource=2.1" \
--data '{
   "resources":[
      "http://www.example.com:9090/sample"
   ],
   "application":"iPlanetAMWebAgentService",
   "subject":{
      "ssoToken":"wpU01SaTq4X2x...NDVFMAAlMxAAA.*"
   }
}' \
"https://openam.example.com:8443/openam/json/policies?_action=evaluate"
[
   {
      "resource":"http://www.example.com:9090/sample",
      "actions":{
         "POST":true,
         "GET":true
      },
      "attributes":{

      },
      "advices":{

      },
      "ttl":9223372036854775807
   }
]
```

AM returns that `demo` can perform `POST` and `GET` operations on the resource.

Chapter 10

Customizing Authentication

This chapter describes how to extend AM authentication features by developing custom modules, nodes, and plugins.

Customizing Authentication Trees

Your deployment might require customizing standard authentication tree features.

For information on customizing authentication nodes, see Authentication Node Development Guide.

Creating Post-Authentication Hooks for Trees

This section explains how to create a hook used by a node within an authentication tree. These tree hooks can perform custom processing after an authentication tree has successfully completed and a session created.

AM includes the following authentication tree hooks:

`CreatePersistentCookieJwt`

Used by the `SetPersistentCookieNode` authentication node.

`UpdatePersistentCookieJwt`

Used by the `PersistentCookieDecisionNode` authentication node.

The Core Class of an Authentication Tree Hook

The following example shows the `UpdatePersistentCookieTreehook` class, as used by the Persistent Cookie Decision node:

```
/*
 * CCPL HEADER START
 *
 * This work is licensed under the Creative Commons
 * Attribution-NonCommercial-NoDerivs 3.0 Unported License.
 * To view a copy of this license, visit
 * https://creativecommons.org/licenses/by-nc-nd/3.0/
 * or send a letter to Creative Commons, 444 Castro Street,
 * Suite 900, Mountain View, California, 94041, USA.
 *
 * You can also obtain a copy of the license at legal-notices/CC-BY-NC-ND.txt.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * If applicable, add the following below this CCPL HEADER, with the fields
 * enclosed by brackets "[]" replaced with your own identifying information:
 *      Portions Copyright [yyyy] [name of copyright owner]
 *
 * CCPL HEADER END
 *
 *      Copyright 2018 ForgeRock AS.
 *
 */

package org.forgerock.openam.auth.nodes.treehook;

import java.util.List;
import java.util.concurrent.TimeUnit;

import javax.inject.Inject;

import org.forgerock.guice.core.InjectorHolder;
import org.forgerock.http.protocol.Cookie;
import org.forgerock.http.protocol.Request;
import org.forgerock.http.protocol.Response;
import org.forgerock.openam.auth.node.api.TreeHook;
import org.forgerock.openam.auth.node.api.TreeHookException;
import org.forgerock.openam.auth.nodes.PersistentCookieDecisionNode;
import org.forgerock.openam.auth.nodes.jwt.InvalidPersistentJwtException;
import org.forgerock.openam.auth.nodes.jwt.PersistentJwtStringSupplier;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.google.inject.assistedinject.Assisted;

/**
 * A TreeHook for updating a persistent cookie.
 */
@TreeHook.Metadata(configClass = PersistentCookieDecisionNode.Config.class)  ❶
public class UpdatePersistentCookieTreeHook implements TreeHook {  ❷

    private final Request request;
    private final Response response;
    private final PersistentCookieDecisionNode.Config config;
    private final PersistentJwtStringSupplier persistentJwtStringSupplier;
    private final PersistentCookieResponseHandler persistentCookieResponseHandler;
    private final Logger logger = LoggerFactory.getLogger("amAuth");

    /**
     * The UpdatePersistentCookieTreeHook Constructor.
     *
     * @param request The request.
     * @param response The response.
     * @param config the config for updating the cookie.
     */
    @Inject  ❸
    public UpdatePersistentCookieTreeHook(@Assisted Request request, @Assisted Response response,
                                          @Assisted PersistentCookieDecisionNode.Config config) {
        this.request = request;
        this.response = response;
        this.config = config;
        this.persistentJwtStringSupplier = InjectorHolder.getInstance(PersistentJwtStringSupplier.class);
        this.persistentCookieResponseHandler =
 InjectorHolder.getInstance(PersistentCookieResponseHandler.class);
    }

    @Override
    public void accept() throws TreeHookException {  ❹
        logger.debug("UpdatePersistentCookieTreeHook.accept");
        String orgName = PersistentCookieResponseHandler.getOrgName(response);
        Cookie originalJwt = getJwtCookie(request, config.persistentCookieName());
        if (originalJwt != null) {
            String jwtString;
            try {
                jwtString = persistentJwtStringSupplier.getUpdatedJwt(originalJwt.getValue(), orgName,
                        String.valueOf(config.hmacSigningKey()), config.idleTimeout().to(TimeUnit.HOURS));
            } catch (InvalidPersistentJwtException e) {
                logger.error("Invalid jwt", e);
                throw new TreeHookException(e);
            }

            if (jwtString != null && !jwtString.isEmpty()) {
                persistentCookieResponseHandler.setCookieOnResponse(response, request,
 config.persistentCookieName(),
                        jwtString, originalJwt.getExpires(), config.useSecureCookie(),
 config.useHttpOnlyCookie());
            }
        }
    }

    private Cookie getJwtCookie(Request request, String cookieName) {
        if (request.getCookies().containsKey(cookieName)) {
            List<Cookie> cookies = request.getCookies().get(cookieName);
            for (Cookie cookie : cookies) {
                if (cookie.getName().equals(cookieName)) {
                    return cookie;
                }
            }
        }
        return null;
    }
}
```

Key:

❶ The `@TreeHook.Metadata` annotation.

Before defining the core class, use a Java `@TreeHook.Metadata` annotation to specify the class the tree hook uses for its configuration. Use the `configClass` property to specify the configuration class of the node that will be using the tree hook.

❷ The core class must implement the `TreeHook` interface. For more information, see the TreeHook interface in the *AM 6.5.5 Public API Javadoc*.

❸ AM uses Google's *Guice* dependency injection framework for authentication nodes and tree hooks. Use the `@Inject` annotation to construct a new instance of the tree hook, specifying the configuration interface set up earlier and any other required parameters.

For more information, see the Inject annotation type and the Assisted annotation type in the *Google Guice Javadoc*.

❹ Creating an `Accept` instance. The main logic of a tree hook is handled by the `Accept` function.

Customizing Authentication Chains

Your deployment might require customizing standard authentication chain features. See the following sections for customization examples:

- Creating a Custom Authentication Module
- Using Server-side Authentication Scripts in Authentication Modules
- Creating Post-Authentication Plugins for Chains
- Customizing CTS-Based Session Quota Exhaustion Actions

Creating a Custom Authentication Module

This section shows how to customize authentication with a sample custom authentication module. For deployments with particular requirements not met by existing AM authentication modules, determine whether you can adapt one of the built-in or extension modules for your needs. If not, build the functionality into a custom authentication module.

About the Sample Authentication Module

The sample authentication module prompts for a user name and password to authenticate the user, and handles error conditions. The sample shows how you integrate an authentication module into AM such that you can configure the module through the AM console, and also localize the user interface.

For information on downloading and building AM sample source code, see How do I access and build the sample code provided for OpenAM 12.x, 13.x and AM (All versions)? in the *Knowledge Base*.

Get a local clone so that you can try the sample on your system. In the sources, you find the following files under the `/path/to/openam-samples-external/custom-authentication-module` directory:

`pom.xml`

Apache Maven project file for the module

This file specifies how to build the sample authentication module, and also specifies its dependencies on AM components and on the Java Servlet API.

`src/main/java/org/forgerock/openam/examples/SampleAuth.java`

Core class for the sample authentication module

This class is called by AM to initialize the module and to process authentication. See "The Sample Authentication Logic" for details.

`src/main/java/org/forgerock/openam/examples/SampleAuthPrincipal.java`

Class implementing `java.security.Principal` interface that defines how to map credentials to identities

This class is used to process authentication. See "The Sample Auth Principal" for details.

`src/main/resources/amAuthSampleAuth.properties`

Properties file mapping UI strings to property values

This file makes it easier to localize the UI. See "Sample Auth Properties" for details.

`src/main/resources/amAuthSampleAuth.xml`

Configuration file for the sample authentication service

This file is used when registering the authentication module with AM. See "The Sample Auth Service Configuration" for details.

`src/main/resources/config/auth/default/SampleAuth.xml`

Callback file for deprecated AM classic UI authentication pages

The sample authentication module does not include localized versions of this file. See "Sample Auth Callbacks" for details.

`src/main/java/org/forgerock/openam/examples/SampleAuthPlugin.java`
`src/main/resources/META-INF/services/org.forgerock.openam.plugins.AmPlugin`

These files serve to register the plugin with AM.

The Java class, `SampleAuthPlugin`, implements the `org.forgerock.openam.plugins.AmPlugin` interface. In the sample, this class registers the `SampleAuth` implementation, and the `amAuthSampleAuth` service schema for configuration.

The services file, `org.forgerock.openam.plugins.AmPlugin`, holds the fully qualified class name of the `AmPlugin` that registers the custom implementations. In this case, `org.forgerock.openam.examples.SampleAuthPlugin`.

For an explanation of service loading, see the `ServiceLoader` API specification.

Sample Auth Properties

AM uses a Java properties file per locale to retrieve the appropriate, localized strings for the authentication module.

The following is the Sample Authentication Module properties file, `amAuthSampleAuth.properties`.

```
sampleauth-service-description=Sample Authentication Module
a500=Authentication Level
a501=Service Specific Attribute

sampleauth-ui-login-header=Login
sampleauth-ui-username-prompt=User Name:
sampleauth-ui-password-prompt=Password:

sampleauth-error-1=Error 1 occurred during the authentication
sampleauth-error-2=Error 2 occurred during the authentication
```

Sample Auth Callbacks

AM callbacks XML files are used to build the deprecated classic UI to prompt the user for identity information needed to process the authentication. The document type for a callback XML file is described in `WEB-INF/Auth_Module_Properties.dtd` where AM is deployed.

The value of the `moduleName` property in the callbacks file must match your custom authentication module's class name. Observe that the module name `SampleAuth`, shown in the example below, matches the class name in "The Sample Authentication Logic" [248].

The following is the `SampleAuth.xml` callbacks file.

```
<!DOCTYPE ModuleProperties PUBLIC
 "=//iPlanet//Authentication Module Properties XML Interface 1.0 DTD//EN"
        "jar://com/sun/identity/authentication/Auth_Module_Properties.dtd">

<ModuleProperties moduleName="SampleAuth" version="1.0" >
    <Callbacks length="0" order="1" timeout="600" header="#NOT SHOWN#" />
    <Callbacks length="2" order="2" timeout="600" header="#TO BE SUBSTITUTED#">
        <NameCallback isRequired="true">
            <Prompt>#USERNAME#</Prompt>
        </NameCallback>
        <PasswordCallback echoPassword="false" >
            <Prompt>#PASSWORD#</Prompt>
        </PasswordCallback>
    </Callbacks>
    <Callbacks length="1" order="3" timeout="600" header="#TO BE SUBSTITUTED#"
        error="true" >
        <NameCallback>
            <Prompt>#THE DUMMY WILL NEVER BE SHOWN#</Prompt>
        </NameCallback>
    </Callbacks>
</ModuleProperties>
```

This file specifies three states.

1. The initial state (order="1") is used dynamically to replace the dummy strings shown between hashes (for example, `#USERNAME#`) by the `substituteUIStrings()` method in `SampleAuth.java`.

2. The next state (order="2") handles prompting the user for authentication information.

3. The last state (order="3") has the attribute `error="true"`. If the authentication module state machine reaches this order then the authentication has failed. The `NameCallback` is not used and not displayed to user. AM requires that the callbacks array have at least one element. Otherwise AM does not permit header substitution.

The Sample Authentication Logic

An AM authentication module must extend the `com.sun.identity.authentication.spi.AMLoginModule` abstract class, and must implement the methods shown below.

Tip

The account lockout functionality in AM is triggered by counting invalid password exceptions, rather than invalid login exceptions.

To trigger account lockouts after repeated failed attempts, ensure your modules throw `InvalidPasswordException` exceptions instead of `AuthLoginException` exceptions when appropriate, as per the code below.

See the *ForgeRock Access Management Java SDK API Specification* for reference.

```
public void init(Subject subject, Map sharedState, Map options)

// OpenAM calls the process() method when the user submits authentication
// information. The process() method determines what happens next:
// success, failure, or the next state specified by the order
// attribute in the callbacks XML file.
public int process(Callback[] callbacks, int state) throws LoginException

// OpenAM expects the getPrincipal() method to return an implementation of
// the java.security.Principal interface.
public Principal getPrincipal()
```

AM does not reuse authentication module instances. This means that you can store information specific to the authentication process in the instance.

The implementation, `SampleAuth.java`, is shown below.

```
/**
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS HEADER.
 *
 * Copyright (c) 2011-2016 ForgeRock AS. All Rights Reserved
 *
 * The contents of this file are subject to the terms
 * of the Common Development and Distribution License
 * (the License). You may not use this file except in
 * compliance with the License.
 *
 * You can obtain a copy of the License at legal/CDDLv1.0.txt.
 * See the License for the specific language governing
 * permission and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL
 * Header Notice in each file and include the License file at legal/CDDLv1.0.txt.
 * If applicable, add the following below the CDDL Header,
 * with the fields enclosed by brackets [] replaced by
 * your own identifying information:
 * "Portions Copyrighted [year] [name of copyright owner]"
 *
 */

package org.forgerock.openam.examples;

import java.security.Principal;
import java.util.Map;
import java.util.ResourceBundle;

import javax.security.auth.Subject;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.login.LoginException;

import com.sun.identity.authentication.spi.AMLoginModule;
import com.sun.identity.authentication.spi.AuthLoginException;
import com.sun.identity.authentication.spi.InvalidPasswordException;
import com.sun.identity.authentication.util.ISAuthConstants;
import com.sun.identity.shared.datastruct.CollectionHelper;
import com.sun.identity.shared.debug.Debug;

/**
 * SampleAuth authentication module example.
 *
 * If you create your own module based on this example, you must modify all
 * occurrences of "SampleAuth" in addition to changing the name of the class.
 *
 * Please refer to OpenAM documentation for further information.
 *
 * Feel free to look at the code for authentication modules delivered with
 * OpenAM, as they implement this same API.
 */
public class SampleAuth extends AMLoginModule {

    // Name for the debug-log
    private final static String DEBUG_NAME = "SampleAuth";
    private final static Debug debug = Debug.getInstance(DEBUG_NAME);

    // Name of the resource bundle
    private final static String amAuthSampleAuth = "amAuthSampleAuth";

    // User names for authentication logic
    private final static String USERNAME = "demo";
    private final static String PASSWORD = "changeit";

    private final static String ERROR_1_USERNAME = "test1";
    private final static String ERROR_2_USERNAME = "test2";

    // Orders defined in the callbacks file
    private final static int STATE_BEGIN = 1;
    private final static int STATE_AUTH = 2;
    private final static int STATE_ERROR = 3;

    // Errors properties
    private final static String SAMPLE_AUTH_ERROR_1 = "sampleauth-error-1";
    private final static String SAMPLE_AUTH_ERROR_2 = "sampleauth-error-2";

    private Map<String, String> options;
    private ResourceBundle bundle;
    private Map<String, String> sharedState;

    public SampleAuth() {
        super();
    }


    /**
     * This method stores service attributes and localized properties for later
     * use.
     * @param subject
     * @param sharedState
     * @param options
     */
    @Override
    public void init(Subject subject, Map sharedState, Map options) {

        debug.message("SampleAuth::init");

        this.options = options;
        this.sharedState = sharedState;
        this.bundle = amCache.getResBundle(amAuthSampleAuth, getLoginLocale());
    }

    @Override
    public int process(Callback[] callbacks, int state) throws LoginException {

        debug.message("SampleAuth::process state: {}", state);

        switch (state) {

            case STATE_BEGIN:
                // No time wasted here - simply modify the UI and
                // proceed to next state
                substituteUIStrings();
                return STATE_AUTH;

            case STATE_AUTH:
                // Get data from callbacks. Refer to callbacks XML file.
                NameCallback nc = (NameCallback) callbacks[0];
                PasswordCallback pc = (PasswordCallback) callbacks[1];
                String username = nc.getName();
                String password = String.valueOf(pc.getPassword());

                //First errorstring is stored in "sampleauth-error-1" property.
                if (ERROR_1_USERNAME.equals(username)) {
                    setErrorText(SAMPLE_AUTH_ERROR_1);
                    return STATE_ERROR;
                }

                //Second errorstring is stored in "sampleauth-error-2" property.
                if (ERROR_2_USERNAME.equals(username)) {
                    setErrorText(SAMPLE_AUTH_ERROR_2);
                    return STATE_ERROR;
                }

                if (USERNAME.equals(username) && PASSWORD.equals(password)) {
                    debug.message("SampleAuth::process User '{}' " +
                            "authenticated with success.", username);
                    return ISAuthConstants.LOGIN_SUCCEED;
                }

                throw new InvalidPasswordException("password is wrong",
                        USERNAME);

            case STATE_ERROR:
                return STATE_ERROR;
            default:
                throw new AuthLoginException("invalid state");
        }
    }

    @Override
    public Principal getPrincipal() {
        return new SampleAuthPrincipal(USERNAME);
    }

    private void setErrorText(String err) throws AuthLoginException {
        // Receive correct string from properties and substitute the
        // header in callbacks order 3.
        substituteHeader(STATE_ERROR, bundle.getString(err));
    }

    private void substituteUIStrings() throws AuthLoginException {
        // Get service specific attribute configured in OpenAM
        String ssa = CollectionHelper.getMapAttr(options, "specificAttribute");

        // Get property from bundle
        String new_hdr = ssa + " " +
                bundle.getString("sampleauth-ui-login-header");
        substituteHeader(STATE_AUTH, new_hdr);

        replaceCallback(STATE_AUTH, 0, new NameCallback(
                bundle.getString("sampleauth-ui-username-prompt")));
        replaceCallback(STATE_AUTH, 1, new PasswordCallback(
                bundle.getString("sampleauth-ui-password-prompt"), false));
    }
}
```

The Sample Auth Principal

The implementation, `SampleAuthPrincipal.java`, is shown below.

```
/**
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS HEADER.
 *
 * Copyright (c) 2011-2016 ForgeRock AS. All Rights Reserved
 *
 * The contents of this file are subject to the terms
 * of the Common Development and Distribution License
 * (the License). You may not use this file except in
 * compliance with the License.
 *
 * You can obtain a copy of the License at legal/CDDLv1.0.txt.
 * See the License for the specific language governing
 * permission and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL
 * Header Notice in each file and include the License file at legal/CDDLv1.0.txt.
 * If applicable, add the following below the CDDL Header,
 * with the fields enclosed by brackets [] replaced by
 * your own identifying information:
 * "Portions Copyrighted [year] [name of copyright owner]"
 *
 */

package org.forgerock.openam.examples;

import java.io.Serializable;
import java.security.Principal;

/**
 * SampleAuthPrincipal represents the user entity.
 */
public class SampleAuthPrincipal implements Principal, Serializable {
    private final static String COLON = " : ";

    private final String name;

    public SampleAuthPrincipal(String name) {

        if (name == null) {
            throw new NullPointerException("illegal null input");
        }

        this.name = name;
    }

    /**
     * Return the LDAP username for this SampleAuthPrincipal.
     *
     * @return the LDAP username for this SampleAuthPrincipal
     */
    @Override
    public String getName() {
        return name;
    }

    /**
     * Return a string representation of this SampleAuthPrincipal.
     *
     * @return a string representation of this
     *         TestAuthModulePrincipal.
     */
    @Override
    public String toString() {
        return new StringBuilder().append(this.getClass().getName())
                .append(COLON).append(name).toString();
    }
    /**
     * Compares the specified Object with this SampleAuthPrincipal
     * for equality. Returns true if the given object is also a
     *  SampleAuthPrincipal  and the two SampleAuthPrincipal have
     * the same username.
     *
     * @param o Object to be compared for equality with this
     *          SampleAuthPrincipal.
     * @return true if the specified Object is equal equal to this
     *         SampleAuthPrincipal.
     */
    @Override
    public boolean equals(Object o) {
        if (o == null) {
            return false;
        }

        if (this == o) {
            return true;
        }

        if (!(o instanceof SampleAuthPrincipal)) {
            return false;
        }
        SampleAuthPrincipal that = (SampleAuthPrincipal) o;

        if (this.getName().equals(that.getName())) {
            return true;
        }
        return false;
    }

    /**
     * Return a hash code for this SampleAuthPrincipal.
     *
     * @return a hash code for this SampleAuthPrincipal.
     */
    @Override
    public int hashCode() {
        return name.hashCode();
    }
}
```

The Sample Auth Service Configuration

AM requires that all authentication modules be configured by means of an AM service. At minimum, the service must include an authentication level attribute. Your module can access these configuration attributes in the `options` parameter passed to the `init()` method.

Some observations about the service configuration file follow in the list below.

- The document type for a service configuration file is described in `WEB-INF/sms.dtd` where AM is deployed.
- The service name is derived from the module name. The service name must have the following format:

  - It must start with either `iPlanetAMAuth` or `sunAMAuth`.
  - The module name must follow. The case of the module name must match the case of the class that implements the custom authentication module.
  - It must end with `Service`.

  In the Sample Auth service configuration, the module name is `SampleAuth` and the service name is `iPlanetAMAuthSampleAuthService`.

- The service must have a localized description, retrieved from a properties file.
- The `i18nFileName` attribute in the service configuration holds the default (non-localized) base name of the Java properties file. The `i18nKey` attributes indicate properties keys to string values in the Java properties file.
- The authentication level attribute name must have the following format:
  - It must start with `iplanet-am-auth-`, `sun-am-auth-`, or `forgerock-am-auth-`.
  - The module name must follow, and must appear in lower case if the attribute name starts with `iplanet-am-auth-` or `forgerock-am-auth-`. If the attribute name starts with `sun-am-auth-`, it must exactly match the case of the module name as it appears in the service name.
  - It must end with `-auth-level`.

  In the Sample Auth service configuration, the authentication level attribute name is `iplanet-am-auth-sampleauth-auth-level`.

- The Sample Auth service configuration includes an example `sampleauth-service-specific-attribute`, which can be configured through the AM console.

The service configuration file, `amAuthSampleAuth.xml`, is shown below. Save a local copy of this file, which you use when registering the module.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
   DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS HEADER.

   Copyright (c) 2011-2016 ForgeRock AS.

   The contents of this file are subject to the terms
   of the Common Development and Distribution License
   (the License). You may not use this file except in
   compliance with the License.

   You can obtain a copy of the License at legal/CDDLv1.0.txt.
   See the License for the specific language governing
   permission and limitations under the License.

   When distributing Covered Code, include this CDDL
   Header Notice in each file and include the License file at legal/CDDLv1.0.txt.
   If applicable, add the following below the CDDL Header,
   with the fields enclosed by brackets [] replaced by
   your own identifying information:
   "Portions Copyrighted [year] [name of copyright owner]"
-->
<!DOCTYPE ServicesConfiguration
    PUBLIC "=//iPlanet//Service Management Services (SMS) 1.0 DTD//EN"
    "jar://com/sun/identity/sm/sms.dtd">

<ServicesConfiguration>
 <Service name="iPlanetAMAuthSampleAuthService" version="1.0">
  <Schema
   serviceHierarchy="/DSAMEConfig/authentication/iPlanetAMAuthSampleAuthService"
   i18nFileName="amAuthSampleAuth" revisionNumber="10"
   i18nKey="sampleauth-service-description" resourceName="sample">
   <Organization>
    <!-- Specify resourceName for a JSON-friendly property in the REST SMS -->
    <AttributeSchema name="iplanet-am-auth-sampleauth-auth-level" resourceName="authLevel"
     type="single" syntax="number_range" rangeStart="0" rangeEnd="2147483647"
     i18nKey="a500">
     <DefaultValues>
      <Value>1</Value>
     </DefaultValues>
    </AttributeSchema>

    <!-- No need for resourceName when the name is JSON-compatible -->
    <AttributeSchema name="specificAttribute"
     type="single" syntax="string" validator="no" i18nKey="a501" />

    <!--
     For Auth Modules, the parent Schema element specifies the REST SMS resourceName,
     and the nested SubSchema must have resourceName="USE-PARENT"
    -->
    <SubSchema name="serverconfig" inheritance="multiple" resourceName="USE-PARENT">
     <AttributeSchema name="iplanet-am-auth-sampleauth-auth-level" resourceName="authLevel"
      type="single" syntax="number_range" rangeStart="0" rangeEnd="2147483647"
      i18nKey="a500">
      <DefaultValues>
       <Value>1</Value>
      </DefaultValues>
     </AttributeSchema>

     <!-- No need for a DefaultValues element when the default is blank -->
     <AttributeSchema name="specificAttribute"
      type="single" syntax="string" validator="no" i18nKey="a501" />

    </SubSchema>
   </Organization>
  </Schema>
 </Service>
</ServicesConfiguration>
```

Building and Installing the Sample Auth Module

Build the module with Apache Maven, and install the module in AM.

For information on downloading and building AM sample source code, see How do I access and build the sample code provided for OpenAM 12.x, 13.x and AM (All versions)? in the *Knowledge Base*.

Installing the Module

Installing the sample authentication module consists of copying the `.jar` file to AM's `WEB-INF/lib/` directory, registering the module with AM, and then restarting AM or the web application container where it runs.

1. Copy the sample authentication module `.jar` file to `WEB-INF/lib/` where AM is deployed.

   ```
   $ cp target/custom*.jar /path/to/tomcat/webapps/openam/WEB-INF/lib/
   ```

2. Restart AM or the container in which it runs.

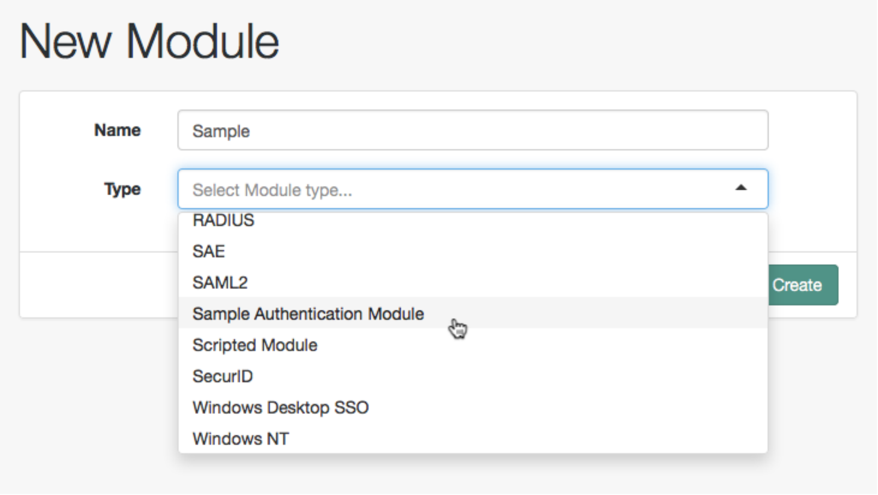
   For example if you deployed AM in Apache Tomcat, then you shut down Tomcat and start it again.

   ```
   $ /path/to/tomcat/bin/shutdown.sh
   $ /path/to/tomcat/bin/startup.sh
   $ tail -1 /path/to/tomcat/logs/catalina.out
   INFO: Server startup in 14736 ms
   ```

Configuring & Testing the Sample Auth Module

Authentication modules are registered as services with AM globally, and then set up for use in a particular realm. In this example, you set up the sample authentication module for use in the realm / (Top Level Realm).

Log in to the AM console as an administrator, such as `amadmin`, and browse to Realms > Top Level Realm > Authentication > Modules. Click Add Module to create an instance of the Sample Authentication Module. Name the module `Sample`.

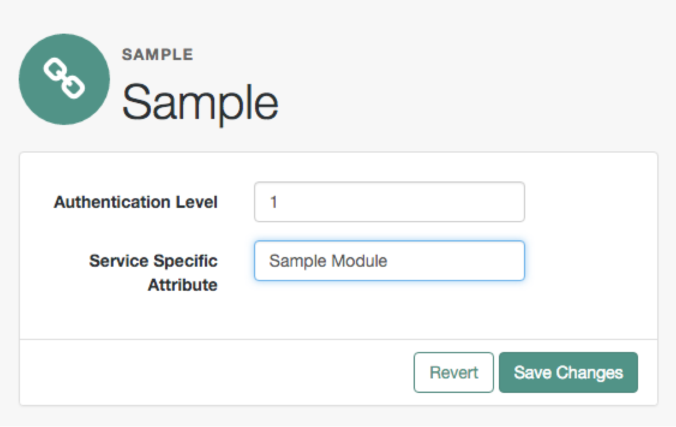


New Module

Name: Sample

Type: Select Module type...

Click Create, and then configure the authentication module as appropriate.



SAMPLE

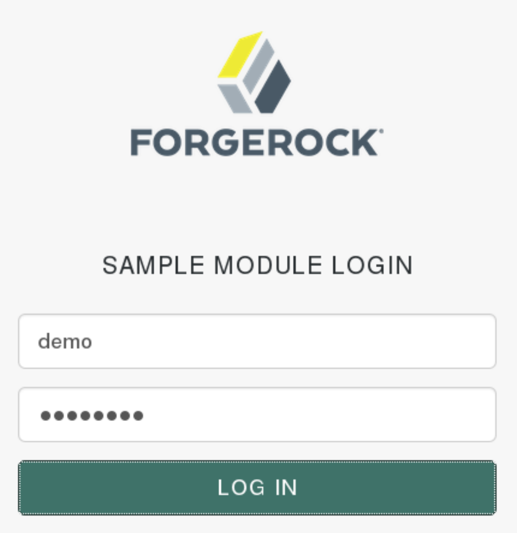
Sample

Authentication Level: 1

Service Specific Attribute: Sample Module

Now that the module is configured, log out of the AM console.

Finally, try the module by specifying the `Sample` module. Browse to the login URL such as `https://openam.example.com:8443/openam/XUI/?realm=/&module=Sample#login`, and then authenticate with user name `demo` and password `changeit`.



SAMPLE MODULE LOGIN

demo

●●●●●●●●

After authentication you are redirected to the end user page for the demo user. You can logout of the AM console, and then try to authenticate as the non-existent user `test123` to see what the error handling looks like to the user.

Using Server-side Authentication Scripts in Authentication Modules

This section demonstrates how to use the default server-side authentication script. An authentication script can be called from a Scripted authentication module.

The default server-side authentication script only authenticates a subject when the current time on the AM server is between 09:00 and 17:00. The script also uses the `logger` and `httpClient` functionality provided in the scripting API.

To examine the contents of the default server-side authentication script in the AM console browse to Realms > Top Level Realm > Scripts, and then click Scripted Module - Server Side.

For general information about scripting in AM, see "*About Scripting*".

For information about APIs available for use when scripting authentication, see the following sections:

- "Global Scripting API Functionality"
- "Authentication API Functionality"

Preparing

AM requires a small amount of configuration before trying the example server-side authentication script. You must create an authentication module of the Scripted type, and then include it in an authentication chain, which can then be used when logging in to AM. You must also ensure the `demo` user has an associated postal address.

The procedures in this section are:

- "To Create a Scripted Authentication Module that Uses the Default Server-side Authentication Script"
- "To Create an Authentication Chain that Uses a Scripted Authentication Module"
- "To Add a Postal Address to the Demo User"

To Create a Scripted Authentication Module that Uses the Default Server-side Authentication Script

In this procedure, create a Scripted Authentication module, and link it to the default server-side authentication script.

1. Log in as an AM administrator, for example `amadmin`.
2. Navigate to Realms > Top Level Realm > Authentication > Modules.
3. On the Authentication Modules page, click Add Module.
4. On the New Module page, enter a module name, such as `myScriptedAuthModule`, from the Type drop-down list, select `Scripted Module`, and then click Create.

5. On the module configuration page:

   a. Uncheck the Client-side Script Enabled checkbox.

   b. From the Server-side Script drop-down list, select `Scripted Module - Server Side`.

   c. Click Save Changes.

To Create an Authentication Chain that Uses a Scripted Authentication Module

In this procedure, create an authentication chain that uses a Data Store authentication module and the Scripted authentication module created in the previous procedure.

1. Log in as an AM administrator, for example `amadmin`.

2. Navigate to Realms > Top Level Realm > Authentication > Chains.

3. On the Authentication Chains page, click Add Chain.

4. On the Add Chain page, enter a name, such as `myScriptedChain`, and then click Create.

5. On the Edit Chain tab, click Add a Module.

6. In the New Module dialog box:

   a. From the Select Module drop-down list, select `DataStore`.

   b. From the Select Criteria drop-down list, select `Required`.

   c. Click OK.

   > **Note**
   >
   > The Data Store authentication module checks the user credentials, whereas the Scripted authentication module does not check credentials, but instead only checks that the authentication request is processed during working hours. Without the Data Store module, the username in the Scripted authentication module cannot be determined. Therefore, do not configure the Scripted authentication module (server-side script) as the *first* module in an authentication chain, because it needs a username.

7. On the Edit Chain tab, click Add Module.

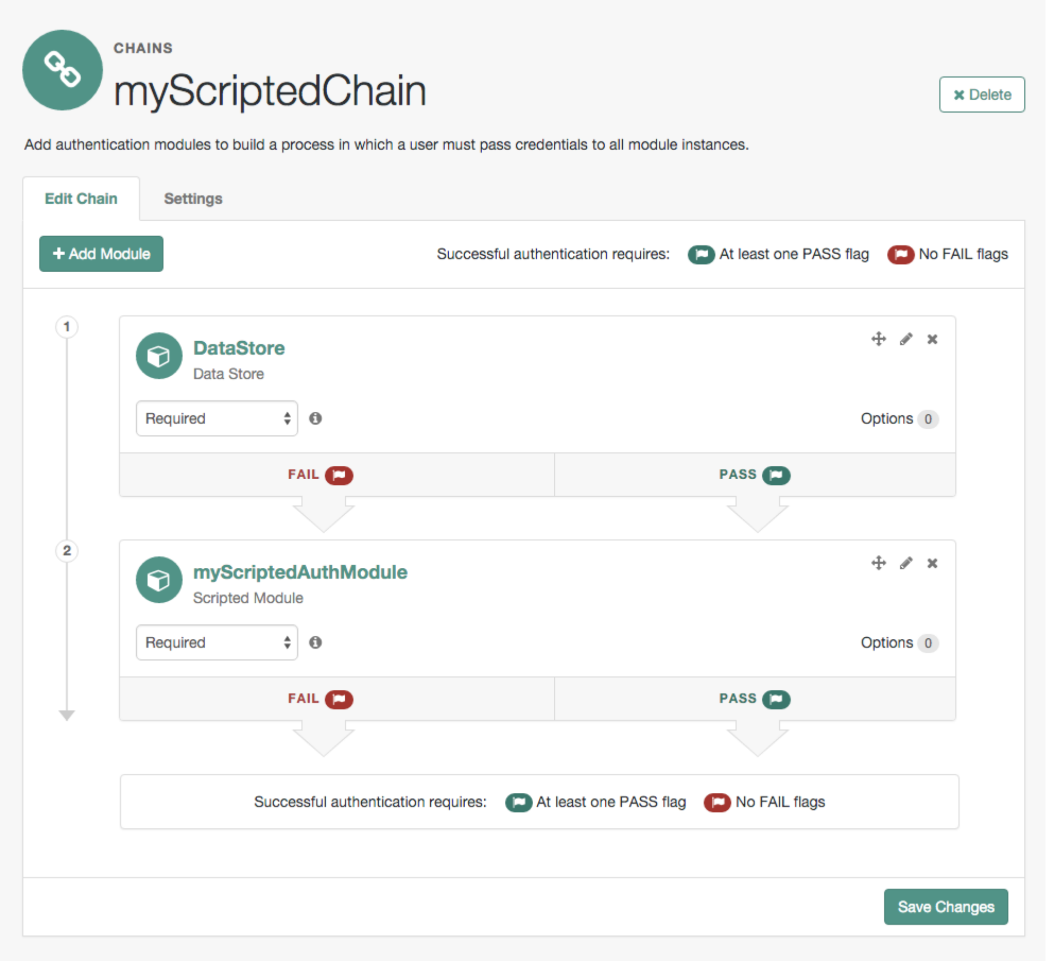
8. In the New Module dialog box:

   a. From the Select Module drop-down list, select the Scripted Module from the previous procedure, for example `myScriptedAuthModule`.

   b. From the Select Criteria drop-down list, select `Required`.

   c. Click OK.

The resulting chain resembles the following:



CHAINS

myScriptedChain

Add authentication modules to build a process in which a user must pass credentials to all module instances.

Successful authentication requires: At least one PASS flag No FAIL flags

1 DataStore
Data Store
Required
Options 0

2 myScriptedAuthModule
Scripted Module
Required
Options 0

Successful authentication requires: At least one PASS flag No FAIL flags

9. On the Edit Chain tab, click Save Changes.

To Add a Postal Address to the Demo User

1. Log in as an AM administrator, for example `amadmin`.
2. Navigate to Realms > Top Level Realm > Identities.
3. On the Identities tab, click the `demo` user.
4. In the Home Address field, enter a valid postal address, with lines separated by commas.

For example:

`ForgeRock Inc., 201 Mission St #2900, San Francisco, CA 94105, USA`

5. Save your changes.

Trying the Default Server-side Authentication Script

This section shows how to log in using an authentication chain that contains a Scripted authentication module, which in turn uses the default server-side authentication script.

The default server-side authentication script gets the postal address of a user after they authenticate using a Data Store authentication module, and then makes an HTTP call to an external web service to determine the longitude and latitude of the address. Using these details, a second HTTP call is performed to get the local time at those coordinates. If that time is between two preset limits, authentication is allowed, and the user is given a session and redirected to the profile page.

To Log in Using a Chain Containing a Scripted Authentication Module

1. Log out of AM.

2. In a browser, navigate to the AM login URL, and specify the authentication chain created in the previous procedure as the value of the `service` parameter.

   For example:

   ```
   https://openam.example.com:8443/openam/XUI/?service=myScriptedChain#login
   ```

3. Log in as user `demo` with password `changeit`.

   If login is successful, the user profile page appears. The script will also output messages, such as the following in the `debug/Authentication` log file:

```
Starting scripted authentication
amScript:02/27/2017 03:22:42:881 PM GMT: Thread[ScriptEvaluator-5,5,main]: TransactionId[7635cd7c-
ea97-4be6-8694-9e2be8642d56-8581]
User: demo
amScript:02/27/2017 03:22:42:882 PM GMT: Thread[ScriptEvaluator-5,5,main]: TransactionId[7635cd7c-
ea97-4be6-8694-9e2be8642d56-8581]
User address: ForgeRock Inc., 201 Mission St #2900, San Francisco, CA 94105, USA
amScript:02/27/2017 03:22:42:929 PM GMT: Thread[ScriptEvaluator-5,5,main]: TransactionId[7635cd7c-
ea97-4be6-8694-9e2be8642d56-8581]
User REST Call. Status: [Status: 200 OK]
amScript:02/27/2017 03:27:31:646 PM GMT: Thread[ScriptEvaluator-7,5,main]: TransactionId[7635cd7c-
ea97-4be6-8694-9e2be8642d56-8581]
latitude:37.7914374 longitude:-122.3950694
amScript:02/27/2017 03:27:31:676 PM GMT: Thread[ScriptEvaluator-7,5,main]: TransactionId[7635cd7c-
ea97-4be6-8694-9e2be8642d56-8581]
User REST Call. Status: [Status: 200 OK]
amScript:02/27/2017 03:27:31:676 PM GMT: Thread[ScriptEvaluator-7,5,main]: TransactionId[7635cd7c-
ea97-4be6-8694-9e2be8642d56-8581]
Current time at the users location: 10
amScript:02/27/2017 03:27:31:676 PM GMT: Thread[ScriptEvaluator-7,5,main]: TransactionId[7635cd7c-
ea97-4be6-8694-9e2be8642d56-8581]
Authentication allowed!
amLoginModule:02/27/2017 03:27:31:676 PM GMT: Thread[http-nio-8080-exec-4,5,main]:
 TransactionId[7635cd7c-ea97-4be6-8694-9e2be8642d56-8581]
Login NEXT State : -1
amLoginModule:02/27/2017 03:27:31:676 PM GMT: Thread[http-nio-8080-exec-4,5,main]:
 TransactionId[7635cd7c-ea97-4be6-8694-9e2be8642d56-8581]
SETTING Module name.... :myScriptedAuthModule
amAuth:02/27/2017 03:27:31:676 PM GMT: Thread[http-nio-8080-exec-4,5,main]: TransactionId[7635cd7c-
ea97-4be6-8694-9e2be8642d56-8581]
Module name is .. myScriptedAuthModule
amAuth:02/27/2017 03:27:31:676 PM GMT: Thread[http-nio-8080-exec-4,5,main]: TransactionId[7635cd7c-
ea97-4be6-8694-9e2be8642d56-8581]
successModuleSet is : [DataStore, myScriptedAuthModule]
amJAAS:02/27/2017 03:27:31:676 PM GMT: Thread[http-nio-8080-exec-4,5,main]: TransactionId[7635cd7c-
ea97-4be6-8694-9e2be8642d56-8581]
login success
```

Tip

The default server-side authentication script outputs log messages at the `message` and `error` level.

AM does not log debug messages from scripts by default. You can configure AM to log such messages by setting the debug log level for the `amScript` service. For details, see "Debug Logging By Service" in the *Setup and Maintenance Guide*.

4. (Optional) To test that the script is being used as part of the login process, edit the script to alter the times when authentication is allowed:

   a. Log out the `demo` user.

   b. Log in as an AM administrator, for example `amadmin`.

   c. Navigate to Realms > Top Level Realm > Scripts > Scripted Module - Server Side.

d. In the script, swap the values for `START_TIME` and `END_TIME`, for example:

```
var START_TIME = 17;
var END_TIME   = 9; //
```

e. Click Save.

f. Repeat steps 1, 2, and 3 above, logging into the module as the `demo` user as before. The authentication result will be the opposite of the previous result, as the allowed times have inverted.

Creating Post-Authentication Plugins for Chains

Post-authentication plugins (PAP) let you include custom processing at the following places in the authentication cycle:

- At the end of the authentication process, immediately before a user is authenticated
- When a user logs out of an AM session

A common use of post-authentication plugins is to set state information in the session object in conjunction with web or Java agents. The post-authentication plugin sets custom session properties, and then the web or Java agent injects the custom properties into the header sent to the protected application.

Two issues should be considered when writing a post-authentication plugin for an AM deployment that uses client-based sessions:

Cookie size

You can set an unlimited number of session properties in a post authentication plugin. When AM creates a client-based session, it writes the session properties into the session cookie, increasing the size of the cookie. Very large session cookies can exceed browser limitations. Therefore, when implementing a post-authentication plugin in a deployment with client-based sessions, be sure to monitor the session cookie size and verify that you have not exceeded browser cookie size limits.

For more information about client-based session cookies, see "Session Cookies".

Cookie security

The AM administrator secures custom session properties in sessions residing in the CTS token store by using firewalls and other typical security techniques.

However, when using client-based sessions, custom session properties are written in cookies and reside on end users' systems. Cookies can be long-lasting and might represent a security issue if any session properties are of a sensitive nature. When developing a post authentication plugin for a deployment that uses client-based sessions, be sure that you are aware of the measures securing the session contained within the cookie.

For more information about client-based session cookie security, see "Configuring Client-Based Session and Authentication Session Security".

This section explains how to create a post-authentication plugin.

Designing Your Post-Authentication Plugin

Your post-authentication plugin class implements the `AMPostAuthProcessInterface` interface, and in particular the following three methods.

```
public void onLoginSuccess(
  Map requestParamsMap,
  HttpServletRequest request,
  HttpServletResponse response,
  SSOToken token
) throws AuthenticationException

public void onLoginFailure(
  Map requestParamsMap,
  HttpServletRequest request,
  HttpServletResponse response
) throws AuthenticationException

public void onLogout(
  HttpServletRequest request,
  HttpServletResponse response,
  SSOToken token
) throws AuthenticationException
```

AM calls the `onLoginSuccess()` and `onLoginFailure()` methods immediately before informing the user of login success or failure, respectively. AM calls the `onLogout()` method only when the user actively logs out, not when a user's session times out. See the *ForgeRock Access Management Java SDK API Specification* for reference.

These methods can perform whatever processing you require. Yet, know that AM calls your methods synchronously as part of the authentication process. Therefore, if your methods take a long time to complete, you will keep users waiting. Minimize the processing done in your post-authentication methods.

Important

Implementing a post-authentication processing plugin in the top level realm can have unexpected effects. OpenAM invokes a post-authentication plugin when the plugin is configured in the top level realm, which will then run for all types of authentication during startup, including user logins and internal administrative logins. The best practice first and foremost is to configure end users to only log into subrealms, while administrators only log into the top level realm. If you need to execute the post-authentication plugin for administrative logins, make sure that the plugin can also handle internal authentications.

An alternate solution is to configure the post-authentication plugin on a per authentication chain basis, which can be configured separately for user logins or internal administrative logins.

Post-authentication plugins must be stateless: they do not maintain state between login and logout. Store any information that you want to save between login and logout in a session property. AM stores session properties in the CTS token store after login, and retrieves them from the token store as part of the logout process.

Building Your Sample Post-Authentication Plugin

The following example post-authentication plugin sets a session property during successful login, writing to its debug log if the operation fails.

```
package com.forgerock.openam.examples;

import java.util.Map;

import com.iplanet.sso.SSOException;
import com.iplanet.sso.SSOToken;

import com.sun.identity.authentication.spi.AMPostAuthProcessInterface;
import com.sun.identity.authentication.spi.AuthenticationException;
import com.sun.identity.shared.debug.Debug;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SamplePAP implements AMPostAuthProcessInterface {
    private final static String PROP_NAME = "MyProperty";
    private final static String PROP_VALUE = "MyValue";
    private final static String DEBUG_FILE = "SamplePAP";

    protected Debug debug = Debug.getInstance(DEBUG_FILE);

    public void onLoginSuccess(
            Map requestParamsMap,
            HttpServletRequest request,
            HttpServletResponse response,
            SSOToken token
    ) throws AuthenticationException {
        try {
            token.setProperty(PROP_NAME, PROP_VALUE);
        } catch (SSOException e) {
            debug.error("Unable to set property");
        }
    }

    public void onLoginFailure(
            Map requestParamsMap,
            HttpServletRequest request,
            HttpServletResponse response
    ) throws AuthenticationException {
        // Not used
    }

    public void onLogout(
            HttpServletRequest request,
            HttpServletResponse response,
            SSOToken token
    ) throws AuthenticationException {
        // Not used
    }
}
```

If you have not already done so, download and build the sample code.

For information on downloading and building AM sample source code, see How do I access and build the sample code provided for OpenAM 12.x, 13.x and AM (All versions)? in the *Knowledge Base*.

In the sources, you find the following files:

`pom.xml`

Apache Maven project file for the module

This file specifies how to build the sample post-authentication plugin, and also specifies its dependencies on AM components and on the Servlet API.

`src/main/java/com/forgerock/openam/examples/SamplePAP.java`

Core class for the sample post-authentication plugin

Once built, copy the .jar to the `WEB-INF/lib` directory where you deployed AM.

```
$ cp target/*.jar /path/to/tomcat/webapps/openam/WEB-INF/lib/
```

Restart AM or the container in which it runs.

Configuring Your Post-Authentication Plugin

You can associate post-authentication plugins with realms or services (authentication chains). Where you configure the plugin depends on the scope to which the plugin should apply:

- Plugins configured at the realm level are executed when authenticating to any authentication chain in the realm, provided the authentication chain does not have an associated plugin.
- Plugins configured at the service level are executed if that authentication chain is used for authentication. Any plugins configured at the realm level will not execute.

In OpenAM Console, navigate to Realms > *Realm Name* > Authentication > Settings > Post Authentication Processing. In the Authentication Post Processing Classes list, add the sample plugin class, `com.forgerock.openam.examples.SamplePAP`, and then click Save.

Alternatively, you can configure sample plugin for the realm by using the **ssoadm** command.

```
$ ssoadm set-svc-attrs \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --servicename iPlanetAMAuthService \
 --realm /myRealm \
 --attributevalues iplanet-am-auth-post-login-process-class=
 com.forgerock.openam.examples.SamplePAP
iPlanetAMAuthService under /myRealm was
 modified.
```

Testing Your Post-Authentication Plugin

To test the sample post-authentication plugin, login successfully to AM in the scope where the plugin is configured. For example, if you configured your plugin for the realm, `/myRealm`, specify the realm in the login URL.

```
https://openam.example.com:8443/openam/XUI/?realm=/myRealm#login
```

Although you will not notice anywhere in the user interface that AM calls your plugin, a web or Java agent or custom client code could retrieve the session property that your plugin added to the user session.

Customizing CTS-Based Session Quota Exhaustion Actions

This section demonstrates a custom session quota exhaustion action plugin. AM calls a session quota exhaustion action plugin when a user tries to open more CTS-based sessions than their quota allows. Note that session quotas are not available for client-based sessions.

You only need a custom session quota exhaustion action plugin if the built-in actions are not flexible enough for your deployment. See "Implementing Session Quotas".

Creating & Installing a Custom Session Quota Exhaustion Action

You build custom session quota exhaustion actions into a .jar that you then plug in to AM. You must also add your new action to the Session service configuration, and restart AM in order to be able to configure it for your use.

Your custom session quota exhaustion action implements the `com.iplanet.dpro.session.service.QuotaExhaustionAction` interface, overriding the `action` method. The `action` method performs the action when the session quota is met, and returns `true` only if the request for a new session should *not* be granted.

The example in this section simply removes the first session it finds as the session quota exhaustion action.

```
package org.forgerock.openam.examples.quotaexhaustionaction;

import static org.forgerock.openam.session.SessionConstants.SESSION_DEBUG;
import com.google.inject.Key;
import com.google.inject.name.Names;
import com.iplanet.dpro.session.SessionException;
import com.iplanet.dpro.session.SessionID;
import com.iplanet.dpro.session.service.InternalSession;
import com.iplanet.dpro.session.service.QuotaExhaustionAction;
import com.sun.identity.shared.debug.Debug;
import org.forgerock.guice.core.InjectorHolder;
import org.forgerock.openam.session.Session;
import org.forgerock.openam.session.clientsdk.SessionCache;

import javax.inject.Inject;
import java.util.Map;

/**
 * This is a sample {@link QuotaExhaustionAction} implementation,
 * which randomly kills the first session it finds.
 */
public class SampleQuotaExhaustionAction implements QuotaExhaustionAction {

    private static Debug debug = InjectorHolder.getInstance(Key.get(Debug.class,
 Names.named(SESSION_DEBUG)));

    private final SessionCache sessionCache;

    public SampleQuotaExhaustionAction() {
        this.sessionCache = InjectorHolder.getInstance(SessionCache.class);
    }

    @Inject
    public SampleQuotaExhaustionAction(SessionCache sessionCache) {
        this.sessionCache = sessionCache;
    }

    /**
     * Check if the session quota for a given user has been exhausted and
     * if so perform the necessary actions. This implementation randomly
     * destroys the first session it finds.
     *
     * @param is               The InternalSession to be activated.
     * @param existingSessions All existing sessions that belong to the same
     *                         uuid (Map:sid->expiration_time).
     * @return true If the session activation request should be rejected,
     *              otherwise false.
     */
    @Override
    public boolean action(
            InternalSession is,
            Map<String, Long> existingSessions) {
        for (Map.Entry<String, Long> entry : existingSessions.entrySet()) {
            try {
                // Get a Session from the cache based on the session ID, and destroy it.
                SessionID sessionId = new SessionID(entry.getKey());
                Session session = sessionCache.getSession(sessionId);
                session.destroySession(sessionId);
                // Only destroy the first session.
                break;
            } catch (SessionException se) {
                if (debug.messageEnabled()) {
                    debug.message("Failed to destroy existing session.", se);
                }
                // In this case, deny the session activation request.
                return true;
            }
        }
        return false;
    }
}
```

If you have not already done so, download and build the sample code.

For information on downloading and building AM sample source code, see How do I access and build the sample code provided for OpenAM 12.x, 13.x and AM (All versions)? in the *Knowledge Base*.

In the sources, you find the following files:

`pom.xml`

Apache Maven project file for the module

This file specifies how to build the sample plugin, and also specifies its dependencies on AM components and on the Servlet API.

`src/main/java/org/forgerock/openam/examples/quotaexhaustionaction/SampleQuotaExhaustionAction.java`

Core class for the sample quota exhaustion action plugin

Once built, copy the .jar to `WEB-INF/lib/` where AM is deployed.

```
$ cp target/*.jar /path/to/tomcat/webapps/openam/WEB-INF/lib/
```

Using the **ssoadm** command, update the Session Service configuration:

```
$ ssoadm \
 set-attr-choicevals \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --servicename iPlanetAMSessionService \
 --schematype Global \
 --attributename iplanet-am-session-constraint-handler \
 --add \
 --choicevalues myKey=\
org.forgerock.openam.examples.quotaexhaustionaction.SampleQuotaExhaustionAction
Choice Values were set.
```

Extract `amSession.properties` and if necessary the localized versions of this file from `openam-core-6.5.5.jar` to `WEB-INF/classes/` where AM is deployed. For example, if AM is deployed under `/path/to/tomcat/webapps/openam`, then you could run the following commands.

```
$ cd /path/to/tomcat/webapps/openam/WEB-INF/classes/
$ jar -xvf ../lib/openam-core-6.5.5.jar amSession.properties
inflated: amSession.properties
```

Add the following line to `amSession.properties`.

```
myKey=Randomly Destroy Session
```

Restart AM or the container in which it runs.

You can now use the new session quota exhaustion action. In the AM console, navigate to Configure > Global Services, click Session, scroll to Resulting behavior if session quota exhausted, and then choose an option.

Before moving to your test and production environments, be sure to add your `.jar` file and updates to `amSession.properties` into a custom `.war` file that you can then deploy. You must still update the Session service configuration in order to use your custom session quota exhaustion action.

Listing Session Quota Exhaustion Actions

List session quota exhaustion actions by using the **ssoadm** command:

```
$ ssoadm \
 get-attr-choicevals \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --servicename iPlanetAMSessionService \
 --schematype Global \
 --attributename iplanet-am-session-constraint-handler

I18n Key                  Choice Value
------------------------- ---...-----------------------------------------
choiceDestroyOldSession   org...session.service.DestroyOldestAction
choiceDenyAccess          org...session.service.DenyAccessAction
choiceDestroyNextExpiring org...session.service.DestroyNextExpiringAction
choiceDestroyAll          org...session.service.DestroyAllAction
myKey                     org...examples...SampleQuotaExhaustionAction
```

Removing a Session Quota Exhaustion Action

Remove a session quota exhaustion action by using the **ssoadm** command:

```
$ ssoadm \
 remove-attr-choicevals \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --servicename iPlanetAMSessionService \
 --schematype Global \
 --attributename iplanet-am-session-constraint-handler \
 --choicevalues \
 org.forgerock.openam.examples.quotaexhaustionaction.SampleQuotaExhaustionAction
Choice Values were removed.
```

Chapter 11
Reference

This reference section covers settings and the scripting API relating to authentication in AM.

Core Authentication Attributes

Every AM realm has a set of authentication properties that applies to all authentication performed to that realm. The settings are referred to as core authentication attributes.

To configure core authentication attributes for an entire AM deployment, navigate to Configure > Authentication in the AM console, and then click Core Attributes.

To override the global core authentication configuration in a realm, navigate to Realms > *Realm Name* > Authentication > Settings in the AM console. Note that when you configure core authentication attributes in a realm, the Global Attributes tab does not appear.

amster service name: `Authentication`

ssoadm service name: `iPlanetAMAuthService`

Global Attributes

The following properties are available under the Global Attributes tab:

Pluggable Authentication Module Classes

Lists the authentication modules classes available to AM. If you have custom authentication modules, add classes to this list that extend from the `com.sun.identity.authentication.spi.AMLoginModule` class.

For more information about custom authentication modules, see "Creating a Custom Authentication Module".

amster attribute: `authenticators`

ssoadm attribute: `iplanet-am-auth-authenticators`

LDAP Connection Pool Size

Sets a minimum and a maximum number of LDAP connections to be used by any authentication module that connects to a specific directory server. This connection pool is different than the SDK connection pool configured in `serverconfig.xml` file.

Format is `host:port:minimum:maximum`.

This attribute is for LDAP and Membership authentication modules only.

amster attribute: `ldapConnectionPoolSize`

ssoadm attribute: `iplanet-am-auth-ldap-connection-pool-size`

Default LDAP Connection Pool Size

Sets the default minimum and maximum number of LDAP connections to be used by any authentication module that connects to any directory server. This connection pool is different than the SDK connection pool configured in `serverconfig.xml` file.

Format is `minimum:maximum`.

When tuning for production, start with 10 minimum, 65 maximum. For example, `10:65`.

This attribute is for LDAP and Membership authentication modules only.

amster attribute: `ldapConnectionPoolDefaultSize`

ssoadm attribute: `iplanet-am-auth-ldap-connection-pool-default-size`

Remote Auth Security

When enabled, AM requires the authenticating application to send its SSO token. This allows AM to obtain the username and password associated with the application.

amster attribute: `remoteAuthSecurityEnabled`

ssoadm attribute: `sunRemoteAuthSecurityEnabled`

Keep Post Process Objects for Logout Processing

When enabled, AM stores instances of post-processing classes into the user session. When the user logs out, the original post-processing classes are called instead of new instances. This may be required for special logout processing.

Enabling this setting increases the memory usage of AM.

amster attribute: `keepPostProcessInstances`

ssoadm attribute: `sunAMAuthKeepPostProcessInstances`

Core

The following properties are available under the Core tab:

Administrator Authentication Configuration

Specifies the default authentication chain used when an administrative user, such as `amAdmin`, logs in to the AM console.

ssoadm attribute:`iplanet-am-auth-admin-auth-module`

Organization Authentication Configuration

Specifies the default authentication chain used when a non-administrative user logs in to AM.

amster attribute: `orgConfig`

ssoadm attribute: `iplanet-am-auth-org-config`

User Profile

The following properties are available under the User Profile tab:

`User Profile`

Specifies whether a user profile needs to exist in the user data store, or should be created on successful authentication. The possible values are:

`true`. Dynamic.

After successful authentication, AM creates a user profile if one does not already exist. AM then issues the SSO token. AM creates the user profile in the user data store configured for the realm.

`createAlias`. Dynamic with User Alias.

After successful authentication, AM creates a user profile that contains the `User Alias List` attribute, which defines one or more aliases for mapping a user's multiple profiles.

`ignore`. Ignored.

After successful authentication, AM issues an SSO token regardless of whether a user profile exists in the data store. The presence of a user profile is not checked.

> **Warning**
>
> Any functionality which needs to map values to profile attributes, such as SAML or OAuth 2.0, will not operate correctly if the User Profile property is set to `ignore`.

`false`. Required.

After successful authentication, the user must have a user profile in the user data store configured for the realm in order for AM to issue an SSO token.

ssoadm attribute: `iplanet-am-auth-dynamic-profile-creation`. Set this attribute's value to one of the following: `true`, `createAlias`, `ignore`, or `false`.

User Profile Dynamic Creation Default Roles

Specifies the distinguished name (DN) of a role to be assigned to a new user whose profile is created when either the `true` or `createAlias` options are selected under the User Profile property. There are no default values. The role specified must be within the realm for which the authentication process is configured.

This role can be either an AM or Sun DSEE role, but it cannot be a filtered role. If you wish to automatically assign specific services to the user, you have to configure the Required Services property in the user profile.

This functionality is deprecated in the *Release Notes*.

amster attribute: `defaultRole`

ssoadm attribute: `iplanet-am-auth-default-role`

Alias Search Attribute Name

After a user is successfully authenticated, the user's profile is retrieved. AM first searches for the user based on the data store settings. If that fails to find the user, AM will use the attributes listed here to look up the user profile. This setting accepts any data store specific attribute name.

amster attribute: `aliasAttributeName`

ssoadm attribute: `iplanet-am-auth-alias-attr-name`

> **Note**
>
> If the `Alias Search Attribute Name` property is empty, AM uses the `iplanet-am-auth-user-naming-attr` property from the `iPlanetAmAuthService`. The `iplanet-am-auth-user-naming-attr` property is only configurable through the **ssoadm** command-line tool and not through the AM console.
>
> ```
> $ ssoadm get-realm-svc-attrs \
> --adminid amadmin \
> --password-file PATH_TO_PWDFILE \
> --realm REALM \
> --servicename iPlanetAMAuthService
>
> $ ssoadm set-realm-svc-attrs \
>  --adminid amadmin \
>  --password-file PATH_TO_PWDFILE \
>  --realm REALM \
>  --servicename iPlanetAMAuthService \
>  --attributevalues iplanet-am-auth-user-naming-attr=SEARCH_ATTRIBUTE
> ```

Account Lockout

The following properties are available under the Account Lockout tab:

Login Failure Lockout Mode

When enabled, AM deactivates the LDAP attribute defined in the Lockout Attribute Name property in the user's profile upon login failure. This attribute works in conjunction with the other account lockout and notification attributes.

amster attribute: `loginFailureLockoutMode`

ssoadm attribute: `iplanet-am-auth-login-failure-lockout-mode`

Login Failure Lockout Count

Defines the number of attempts that a user has to authenticate within the time interval defined in Login Failure Lockout Interval before being locked out.

amster attribute: `loginFailureCount`

ssoadm attribute: `iplanet-am-auth-login-failure-count`

Login Failure Lockout Interval

Defines the time in minutes during which failed login attempts are counted. If one failed login attempt is followed by a second failed attempt within this defined lockout interval time, the lockout count starts, and the user is locked out if the number of attempts reaches the number defined by the Login Failure Lockout Count property. If an attempt within the defined lockout interval time proves successful before the number of attempts reaches the number defined by the Login Failure Lockout Count property, the lockout count is reset.

amster attribute: `loginFailureDuration`

ssoadm attribute: `iplanet-am-auth-login-failure-duration`

Email Address to Send Lockout Notification

Specifies one or more email addresses to which notification is sent if a user lockout occurs.

Separate multiple addresses with spaces, and append `|locale|charset` to addresses for recipients in non-English locales.

amster attribute: `lockoutEmailAddress`

ssoadm attribute: `iplanet-am-auth-lockout-email-address`

Warn User After N Failures

Specifies the number of authentication failures after which AM displays a warning message that the user will be locked out.

ssoadm attribute: `iplanet-am-auth-lockout-warn-user`

Login Failure Lockout Duration

Defines how many minutes a user must wait after a lockout before attempting to authenticate again. Entering a value greater than 0 enables memory lockout and disables physical lockout. *Memory lockout* means the user's account is locked in memory for the number of minutes specified. The account is unlocked after the time period has passed.

amster attribute: `lockoutDuration`

ssoadm attribute: `iplanet-am-auth-lockout-duration`

Lockout Duration Multiplier

Defines a value with which to multiply the value of the Login Failure Lockout Duration attribute for each successive lockout. For example, if Login Failure Lockout Duration is set to 3 minutes, and the Lockout Duration Multiplier is set to 2, the user is locked out of the account for 6 minutes. After the 6 minutes has elapsed, if the user again provides the wrong credentials, the lockout duration is then 12 minutes. With the Lockout Duration Multiplier, the lockout duration is incrementally increased based on the number of times the user has been locked out.

amster attribute: `lockoutDurationMultiplier`

ssoadm attribute: `sunLockoutDurationMultiplier`

Lockout Attribute Name

Defines the LDAP attribute used for physical lockout. The default attribute is `inetuserstatus`, although the field in the AM console is empty. The Lockout Attribute Value field must also contain an appropriate value.

amster attribute: `lockoutAttributeName`

ssoadm attribute: `iplanet-am-auth-lockout-attribute-name`

Lockout Attribute Value

Specifies the action to take on the attribute defined in Lockout Attribute Name. The default value is `inactive`, although the field in the AM console is empty. The Lockout Attribute Name field must also contain an appropriate value.

amster attribute: `lockoutAttributeValue`

ssoadm attribute: `iplanet-am-auth-lockout-attribute-value`

Invalid Attempts Data Attribute Name

Specifies the LDAP attribute used to hold the number of failed authentication attempts towards Login Failure Lockout Count. Although the field appears empty in the AM console, AM stores this data in the `sunAMAuthInvalidAttemptsDataAttrName` attribute defined in the `sunAMAuthAccountLockout` objectclass by default.

amster attribute: `invalidAttemptsDataAttributeName`

ssoadm attribute: `sunAMAuthInvalidAttemptsDataAttrName`

Store Invalid Attempts in Data Store

When enabled, AM stores the information regarding failed authentication attempts as the value of the Invalid Attempts Data Attribute Name in the user data store. Information stored includes number of invalid attempts, time of last failed attempt, lockout time and lockout duration. Storing this information in the identity repository allows it to be shared among multiple instances of AM.

Enable this property to track invalid log in attempts when using CTS-based or client-based authentication sessions.

amster attribute: `storeInvalidAttemptsInDataStore`

ssoadm attribute: `sunStoreInvalidAttemptsInDS`

General

The following properties are available under the General tab:

Default Authentication Locale

Specifies the default language subtype to be used by the Authentication Service. The default value is `en_US`.

amster attribute: `locale`

ssoadm attribute: `iplanet-am-auth-locale`

Identity Types

Lists the type or types of identities used during a profile lookup. You can choose more than one to search on multiple types if you would like AM to conduct a second lookup if the first lookup fails. The possible values are:

`Agent`

Searches for identities under your agents.

`agentgroup`

Searches for identities according to your established agent group.

`agentonly`

Searches for identities only under your agents.

`Group`

Searches for identities according to your established groups.

`User`

Searches for identities according to your users.

Default: `Agent` and `User`.

amster attribute: `identityType`

ssoadm attribute: `sunAMIdentityType`

Pluggable User Status Event Classes

Specifies one or more Java classes used to provide a callback mechanism for user status changes during the authentication process. The Java class must implement the `com.sun.identity.authentication.spi.AMAuthCallBack` interface. AM supports account lockout and password changes. AM supports password changes through the LDAP authentication module, and so the feature is only available for the LDAP module.

A `.jar` file containing the user status event class belongs in the `WEB-INF/lib` directory of the deployed AM instance. If you do not build a `.jar` file, add the class files under `WEB-INF/classes`.

amster attribute: `userStatusCallbackPlugins`

ssoadm attribute: `sunAMUserStatusCallbackPlugins`

Use Client-Based Sessions

When enabled, AM assigns *client-based* sessions to users authenticating to this realm. Otherwise, AM users authenticating to this realm are assigned *CTS-based* sessions.

For more information about sessions, see "About Sessions" [15].

amster attribute: `statelessSessionsEnabled`

ssoadm attribute: `openam-auth-stateless-sessions`

Two Factor Authentication Mandatory

When enabled, users authenticating to a chain that includes a ForgeRock Authenticator (OATH) module are always required to perform authentication using a registered device before they can access AM. When not selected, users can opt to forego registering a device and providing a token and still successfully authenticate.

Letting users choose not to provide a verification token while authenticating carries implications beyond the `required`, `optional`, `requisite`, or `sufficient` flag settings on the ForgeRock Authenticator

(OATH) module in the authentication chain. For example, suppose you configured authentication as follows:

- The ForgeRock Authenticator (OATH) module is in an authentication chain.
- The ForgeRock Authenticator (OATH) module has the `required` flag set.
- Two Factor Authentication Mandatory is not selected.

Users authenticating to the chain can authenticate successfully *without* providing tokens from their devices. The reason for successful authentication in this case is that the `required` setting relates to the execution of the ForgeRock Authenticator (OATH) module itself. Internally, the ForgeRock Authenticator (OATH) module has the ability to forego processing a token while still returning a passing status to the authentication chain.

> **Note**
>
> The `Two Factor Authentication Mandatory` property only applies to modules within authentication chains, and does not affect nodes within authentication trees.

amster attribute: `twoFactorRequired`

ssoadm attribute: `forgerockTwoFactorAuthMandatory`

Default Authentication Level

Specifies the default authentication level for authentication modules.

amster attribute: `defaultAuthLevel`

ssoadm attribute: `iplanet-am-auth-default-auth-level`

Trees

The following properties are available under the Trees tab:

Authentication session state management scheme

Specifies the location where AM stores authentication sessions.

Possible values are:

- `CTS`. AM stores authentication sessions in the CTS token store.
- `JWT`. AM sends the authentication session to the client as a JWT.
- `In-Memory`. AM stores authentication sessions in its memory.

For more information on authentication session storage locations, and the requirements for each, see "Session Storage Location".

Default: `JWT` (new installations), `In-Memory` (after upgrade)

amster attribute: `authenticationSessionsStateManagement`

ssoadm attribute: `openam-auth-authentication-sessions-state-management-scheme`

Max duration (minutes)

Specifies the duration of the authentication session in minutes.

Default: `5`

amster attribute: `authenticationSessionsMaxDuration`

ssoadm attribute: `openam-auth-authentication-sessions-max-duration`

Enable whitelisting

When enabled, AM whitelists authentication sessions to protect them against replay attacks.

Default: Disabled

amster attribute: `authenticationSessionsWhitelist`

ssoadm attribute: `openam-auth-authentication-sessions-whitelist`

Security

The following properties are available under the Security tab:

Module Based Authentication

When enabled, users can authenticate using module-based authentication. Otherwise, all attempts at authentication using the `module=`*`module-name`* login parameter result in failure.

ForgeRock recommends disabling module-based authentication in production environments.

amster attribute: `moduleBasedAuthEnabled`

ssoadm attribute: `sunEnableModuleBasedAuth`

Persistent Cookie Encryption Certificate Alias

Specifies the key pair alias in the AM keystore to use for encrypting persistent cookies.

Default: `test`

amster attribute: `keyAlias`

ssoadm attribute: `iplanet-am-auth-key-alias`

Zero Page Login

When enabled, AM allows users to authenticate using only GET request parameters without showing a login screen.

> **Caution**
>
> Enable with caution as browsers can cache credentials and servers can log credentials when they are part of the URL.

AM always allows HTTP POST requests for zero page login.

Default: false (disabled)

amster attribute: `zeroPageLoginEnabled`

ssoadm attribute: `openam.auth.zero.page.login.enabled`

Zero Page Login Referer Whitelist

Lists the HTTP referer URLs for which AM allows zero page login. These URLs are supplied in the `Referer` HTTP request header, allowing clients to specify the web page that provided the link to the requested resource.

When zero page login is enabled, including the URLs for the pages from which to allow zero page login will provide some mitigation against Login Cross-Site Request Forgery (CSRF) attacks. Leave this list blank to allow zero page login from any Referer.

This setting applies for both HTTP GET and also HTTP POST requests for zero page login.

amster attribute: `zeroPageLoginReferrerWhiteList`

ssoadm attribute: `openam.auth.zero.page.login.referer.whitelist`

Zero Page Login Allowed Without Referer?

When enabled, allows zero page login for requests without an HTTP `Referer` request header. Zero page login must also be enabled.

Enabling this setting reduces the risk of login CSRF attacks with zero page login enabled, but may potentially deny legitimate requests.

amster attribute: `zeroPageLoginAllowedWithoutReferrer`

ssoadm attribute: `openam.auth.zero.page.login.allow.null.referer`

Organization Authentication Signing Secret

Specifies a cryptographically-secure random-generated HMAC shared secret for signing RESTful authentication requests. When users attempt to authenticate to the XUI, AM signs a JSON Web Token (JWT) containing this shared secret. The JWT contains the authentication session ID, realm, and authentication index type value, but does *not* contain the user's credentials.

When modifying this value, ensure the new shared secret is Base-64 encoded and at least 128 bits in length.

amster attribute: `sharedSecret`

ssoadm attribute: `iplanet-am-auth-hmac-signing-shared-secret`

Post Authentication Processing

The following properties are available under the Post Authentication Processing tab:

Default Success Login URL

Accepts a list of values that specifies where users are directed after successful authentication. The format of this attribute is *`client-type|URL`* although the only value you can specify at this time is a URL which assumes the type HTML. The default value is `/openam/console`. Values that do not specify HTTP have that appended to the deployment URI.

amster attribute: `loginSuccessUrl`

ssoadm attribute: `iplanet-am-auth-login-success-url`

Default Failure Login URL

Accepts a list of values that specifies where users are directed after authentication has failed. The format of this attribute is *`client-type|URL`* although the only value you can specify at this time is a URL which assumes the type HTML. Values that do not specify HTTP have that appended to the deployment URI.

amster attribute: `loginFailureUrl`

ssoadm attribute: `iplanet-am-auth-login-failure-url`

Authentication Post Processing Classes

Specifies one or more Java classes used to customize post authentication processes for successful or unsuccessful logins. The Java class must implement the `com.sun.identity.authentication.spi.AMPostAuthProcessInterface` AM interface.

A `.jar` file containing the post processing class belongs in the `WEB-INF/lib` directory of the deployed AM instance. If you do not build a `.jar` file, add the class files under `WEB-INF/classes`. For deployment, add the `.jar` file or classes into a custom AM `.war` file.

For information on creating post-authentication plugins, see "Creating Post-Authentication Plugins for Chains".

amster attribute: `loginPostProcessClass`

ssoadm attribute: `iplanet-am-auth-post-login-process-class`

Generate UserID Mode

When enabled, the Membership module generates a list of alternate user identifiers if the one entered by a user during the self-registration process is not valid or already exists. The user IDs are generated by the class specified in the Pluggable User Name Generator Class property.

amster attribute: `usernameGeneratorEnabled`

ssoadm attribute: `iplanet-am-auth-username-generator-enabled`

Pluggable User Name Generator Class

Specifies the name of the class used to generate alternate user identifiers when Generate UserID Mode is enabled. The default value is `com.sun.identity.authentication.spi.DefaultUserIDGenerator`.

amster attribute: `usernameGeneratorClass`

ssoadm attribute: `iplanet-am-auth-username-generator-class`

User Attribute Mapping to Session Attribute

Enables the authenticating user's identity attributes (stored in the identity repository) to be set as session properties in the user's SSO token. The value takes the format *`User-Profile-Attribute|Session-Attribute-Name`*. If *Session-Attribute-Name* is not specified, the value of *User-Profile-Attribute* is used. All session attributes contain the `am.protected` prefix to ensure that they cannot be edited by the client applications.

For example, if you define the user profile attribute as `mail` and the user's email address, available in the user session, as `user.mail`, the entry for this attribute would be `mail|user.mail`. After a successful authentication, the `SSOToken.getProperty(String)` method is used to retrieve the user profile attribute set in the session. The user's email address is retrieved from the user's session using the `SSOToken.getProperty("am.protected.user.mail")` method call.

Properties that are set in the user session using User Attribute Mapping to Session Attributes cannot be modified (for example, `SSOToken.setProperty(String, String)`). This results in an `SSOException`. Multivalued attributes, such as `memberOf`, are listed as a single session variable with a `|` separator.

When configuring authentication for a realm configured for client-based sessions, be careful not to add so many session attributes that the session cookie size exceeds the maximum allowable cookie size. For more information about client-based session cookies, see "Session Cookies".

amster attribute: `userAttributeSessionMapping`

ssoadm attribute: `sunAMUserAttributesSessionMapping`

> **Important**
>
> The use of this property does not apply to authentication trees. As of AM 6.5.3, use the Scripted Decision Node to retrieve user attributes and session properties, or the Set Session Properties Node for session properties only.

Authentication Module Properties

This section provides a reference to configuration properties for AM authentication modules.

Account Active Check Module

Lets you determine whether an account is marked as active, or locked.

By default, AM checks if a user account is active or locked after processing an entire authentication chain. This means users with locked accounts may be asked to perform unnecessary authentication steps, such as providing a one-time password, before authentication fails.

Use the Account Active Check module to check for active or locked status immediately after determining the user account; for example, after a DataStore or LDAP module. If the account is locked, the chain will fail early, without processing modules that appear after the Account Active Check module.

For more information, see "*Implementing Account Lockout*".

Active Directory Module Properties

amster service name: `ActiveDirectoryModule`

ssoadm service name: `sunAMAuthADService`

Primary Active Directory Server
Secondary Active Directory Server

Specify the primary and secondary Active Directory server(s). AM attempts to contact the primary server(s) first, If no primary server is available, then AM attempts to contact the secondary server(s).

When authenticating users from a directory server that is remote to AM, set the primary server values, and optionally the secondary server values. Primary servers have priority over secondary servers.

To allow users to change passwords through AM, Active Directory requires that you connect over SSL. The default port for LDAP is 389. If you are connecting to Active Directory over SSL, the default port for LDAP/SSL is 636.

For SSL or TLS security, enable the SSL/TLS Access to Active Directory Server property. Make sure that AM can trust the Active Directory certificate when using this option.

ssoadm attributes are: primary is `iplanet-am-auth-ldap-server`; secondary is `iplanet-am-auth-ldap-server2`.

Both properties may take a single value in the form of `server:port`, or more than one value in the form of `openam_full_server_name | server:port`; thus, allowing more than one primary or secondary remote server, respectively.

Assuming a multi-data center environment, AM determines priority within the primary and secondary remote servers as follows:

- Every LDAP server that is mapped to the current AM instance has highest priority.

  For example, if you are connected to `openam1.example.com` and `ldap1.example.com` is mapped to that AM instance, then AM uses `ldap1.example.com`.

- Every LDAP server that was not specifically mapped to a given AM instance has the next highest priority.

  For example, if you have another LDAP server, `ldap2.example.com`, that is not connected to a specific AM server and if `ldap1.example.com` is unavailable, AM connects to the next highest priority LDAP server, `ldap2.example.com`.

- LDAP servers that are mapped to different AM instances have the lowest priority.

  For example, if `ldap3.example.com` is connected to `openam3.example.com` and `ldap1.example.com` and `ldap2.example.com` are unavailable, then `openam1.example.com` connects to `ldap3.example.com`.

DN to Start User Search

Specifies the base DN from which AM searches for users to authenticate.

LDAP data is organized hierarchically, a bit like a file system on Windows or UNIX. More specific DNs likely result in better performance. When configuring the module for a particular part of the organization, you can perhaps start searches from a specific organizational unit, such as `OU=sales,DC=example,DC=com`.

If multiple entries exist with identical search attribute values, make this value specific enough to return only one entry.

amster attribute: `userSearchStartDN`

ssoadm attribute: `iplanet-am-auth-ldap-base-dn`

Bind User DN, Bind User Password

Specify the user and password to authenticate to Active Directory.

If AM stores attributes in Active Directory, for example to manage account lockout, or if Active Directory requires that AM authenticate in order to read users' attributes, then AM needs the DN and password to authenticate to Active Directory.

If the administrator authentication chain (default: `ldapService`) has been configured to include only the Active Directory module, then make sure that the password is correct before you logout. If it is incorrect, you will be locked out. If you do get locked out, you can login with the superuser DN, which by default is `uid=amAdmin,ou=People,`*`AM-deploy-base`*, where *AM-deploy-base* was set during AM configuration.

ssoadm attributes: `iplanet-am-auth-ldap-bind-dn` and `iplanet-am-auth-ldap-bind-passwd`

Attribute Used to Retrieve User Profile
Attributes Used to Search for a User to be Authenticated
User Search Filter
Search Scope

LDAP searches for user entries with attribute values matching the filter you provide. For example, if you search under `CN=Users,DC=example,DC=com` with a filter `"(MAIL=bjensen@example.com)"`, then the directory returns the entry that has `MAIL=bjensen@example.com`. In this example the attribute used to search for a user is `mail`. Multiple attribute values mean the user can authenticate with any one of the values. For example, if you have both `uid` and `mail`, then Barbara Jensen can authenticate with either `bjensen` or `bjensen@example.com`.

The User Search Filter text box provides a more complex filter. For example, if you search on `mail` and add User Search Filter `(objectClass=inetOrgPerson)`, then AM uses the resulting search filter `(&(mail=`*`address`*`) (objectClass=inetOrgPerson))`, where *address* is the mail address provided by the user.

This controls how and the level of the directory that will be searched. You can set the search to run at a high level or against a specific area:

- OBJECT will search only for the entry specified as the DN to Start User Search.
- ONELEVEL will search only the entries that are directly children of that object.
- SUBTREE will search the entry specified and every entry under it.

ssoadm attributes: `iplanet-am-auth-ldap-user-naming-attribute`, `iplanet-am-auth-ldap-user-search-attributes`, `iplanet-am-auth-ldap-search-filter`, and `iplanet-am-auth-ldap-search-scope`

LDAP Connection Mode

If you want to initiate secure communications to data stores using SSL or StartTLS, AM must be able to trust Active Directory certificates, either because the Active Directory certificates were signed by a CA whose certificate is already included in the trust store used by the container where AM runs, or because you imported the certificates into the trust store.

ssoadm attribute: `openam-auth-ldap-connection-mode`

Possible values: `LDAP`, `LDAPS`, and `StartTLS`

Return User DN to DataStore

When enabled, and AM uses Active Directory as the user store, the module returns the DN rather than the User ID, so the bind for authentication can be completed without a search to retrieve the DN.

amster attribute: `returnUserDN`

ssoadm attribute: `iplanet-am-auth-ldap-return-user-dn`

User Creation Attributes

Maps internal attribute names used by AM to external attribute names from Active Directory for dynamic profile creation. Values are of the format `internal_attr1|external_attr1`.

amster attribute: `profileAttributeMappings`

ssoadm attribute: `iplanet-am-ldap-user-creation-attr-list`

Trust All Server Certificates

When enabled, the module trusts all server certificates, including self-signed certificates.

amster attribute: `trustAllServerCertificates`

ssoadm attribute: `iplanet-am-auth-ldap-ssl-trust-all`

LDAP Connection Heartbeat Interval

Specifies how often AM should send a heartbeat request to the directory server to ensure that the connection does not remain idle. Some network administrators configure firewalls and load balancers to drop connections that are idle for too long. You can turn this off by setting the value to 0 or to a negative number. To set the units for the interval, use LDAP Connection Heartbeat Time Unit.

Default: 1

amster attribute: `connectionHeartbeatInterval`

ssoadm attribute: `openam-auth-ldap-heartbeat-interval`

LDAP Connection Heartbeat Time Unit

Specifies the time unit corresponding to LDAP Connection Heartbeat Interval. Possible values are `SECONDS`, `MINUTES`, and `HOURS`.

amster attribute: `connectionHeartbeatTimeUnit`

ssoadm attribute: `openam-auth-ldap-heartbeat-timeunit`

LDAP operations timeout

Defines the timeout in milliseconds that AM should wait for a response from the directory server.

Default: 0 (means no timeout)

amster attribute: `operationTimeout`

ssoadm attribute: `openam-auth-ldap-operation-timeout`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

amster attribute: `authenticationLevel`

ssoadm attribute: `sunAMAuthADAuthLevel`

Adaptive Risk Authentication Module Properties

amster service name: `AdaptiveRiskModule`

ssoadm service name: `sunAMAuthAdaptiveService`

General

The following properties are available under the General tab:

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

amster attribute: `authenticationLevel`

ssoadm attribute: `openam-auth-adaptive-auth-level`

Risk Threshold

Sets the risk threshold score. If the sum of the scores is greater than the threshold, the Adaptive Risk module fails.

Default: 1

amster attribute: `riskThreshold`

ssoadm attribute: `openam-auth-adaptive-auth-threshold`

Failed Authentications

The following properties are available under the Failed Authentications tab:

Failed Authentication Check

When enabled, checks the user profile for authentication failures since the last successful login. This check therefore requires AM to have access to the user profile, and Account Lockout to be enabled (otherwise, AM does not record authentication failures).

amster attribute: `failedAuthenticationCheckEnabled`

ssoadm attribute: `openam-auth-adaptive-failure-check`

Score

Sets the value to add to the total score if the user fails the Failed Authentication Check. Default: 1

amster attribute: `failureScore`

ssoadm attribute: `openam-auth-adaptive-failure-score`

Invert Result

When enabled, adds the score to the total score if the user passes the Failed Authentication Check.

amster attribute: `invertFailureScore`

ssoadm attribute: `openam-auth-adaptive-failure-invert`

IP Address Range

The following properties are available under the IP Address Range tab:

IP Range Check

When enabled, checks whether the client IP address is within one of the specified IP Ranges.

amster attribute: `ipRangeCheckEnabled`

ssoadm attribute: `openam-auth-adaptive-ip-range-check`

IP Range

For IPv4, specifies a list of IP ranges either in CIDR-style notation (`x.x.x.x/YY`) or as a range from one address to another (`x.x.x.x-y.y.y.y`, meaning from *x.x.x.x* to *y.y.y.y*).

For IPv6, specifies a list of IP ranges either in CIDR-style notation (`X:X:X:X:X:X:X:X/YY`) or as a range from one address to another (`X:X:X:X:X:X:X:X-Y:Y:Y:Y:Y:Y:Y:Y`, (`X:X:X:X:X:X:X:X-Y:Y:Y:Y:Y:Y:Y:Y`, meaning from *X:X:X:X:X:X:X:X* to *Y:Y:Y:Y:Y:Y:Y:Y*).

amster attribute: `ipRange`

ssoadm attribute: `openam-auth-adaptive-ip-range-range`

Score

Sets the value to add to the total score if the user fails the IP Range Check.

amster attribute: `ipRangeScore`

ssoadm attribute: `openam-auth-adaptive-ip-range-score`

Invert Result

When enabled, adds the Score to the total score if the user passes the IP Range Check.

amster attribute: `invertIPRangeScoreEnabled`

ssoadm attribute: `openam-auth-adaptive-ip-range-invert`

IP Address History

The following properties are available under the IP Address History tab:

IP History Check

When enabled, checks whether the client IP address matches one of the known values stored on the profile attribute you specify. This check therefore requires that AM have access to the user profile.

amster attribute: `ipHistoryCheckEnabled`

ssoadm attribute: `openam-auth-adaptive-ip-history-check`

History size

Specifies how many IP address values to retain on the profile attribute you specify.

Default: 5

amster attribute: `ipHistoryCount`

ssoadm attribute: `openam-auth-ip-adaptive-history-count`

Profile Attribute Name

Specifies the name of the user profile attribute in which to store known IP addresses. Ensure the specified attribute exists in your user data store; the `iphistory` attribute does not exist by default, and it is not created when performing AM schema updates.

Default: `iphistory`

amster attribute: `ipHistoryProfileAttribute`

ssoadm attribute: `openam-auth-adaptive-ip-history-attribute`

Save Successful IP Address

When enabled, saves new client IP addresses to the known IP address list following successful authentication.

amster attribute: `saveSuccessfulIP`

ssoadm attribute: `openam-auth-adaptive-ip-history-save`

Score

Sets the value to add to the total score if the user fails the IP History Check.

Default: 1

amster attribute: `ipHistoryScore`

ssoadm attribute: `openam-auth-adaptive-ip-history-score`

Invert Result

When enabled, adds the Score to the total score if the user passes the IP History Check.

amster attribute: `invertIPHistoryScore`

ssoadm attribute: `openam-auth-adaptive-ip-history-invert`

Known Cookie

The following properties are available under the Known Cookie tab:

Cookie Value Check

When enabled, checks whether the client browser request has the specified cookie and optional cookie value.

amster attribute: `knownCookieCheckEnabled`

ssoadm attribute: `openam-auth-adaptive-known-cookie-check`

Cookie Name

Specifies the name of the cookie for which AM checks when you enable the Cookie Value Check.

amster attribute: `knownCookieName`

ssoadm attribute: `openam-auth-adaptive-known-cookie-name`

Cookie Value

Specifies the value of the cookie for which AM checks. If no value is specified, AM does not check the cookie value.

amster attribute: `knownCookieValue`

ssoadm attribute: `openam-auth-adaptive-known-cookie-value`

Save Cookie Value on Successful Login

When enabled, saves the cookie as specified in the client's browser following successful authentication. If no Cookie Value is specified, the value is set to 1.

amster attribute: `createKnownCookieOnSuccessfulLogin`

ssoadm attribute: `openam-auth-adaptive-known-cookie-save`

Score

Sets the value to add to the total score if user passes the Cookie Value Check.

Default: 1

amster attribute: `knownCookieScore`

ssoadm attribute: `openam-auth-adaptive-known-cookie-score`

Invert Result

When enabled, adds the Score to the total score if the user passes the Cookie Value Check.

amster attribute: `invertKnownCookieScore`

ssoadm attribute: `openam-auth-adaptive-known-cookie-invert`

Device Cookie

The following properties are available under the Device Cookie tab:

Device Registration Cookie Check

When enabled, the cookie check passes if the client request contains the cookie specified in Cookie Name.

amster attribute: `deviceCookieCheckEnabled`

ssoadm attribute: `openam-auth-adaptive-device-cookie-check`

Cookie Name

Specifies the name of the cookie for the Device Registration Cookie Check.

Default: Device

amster attribute: `deviceCookieName`

ssoadm attribute: `openam-auth-adaptive-device-cookie-name`

Save Device Registration on Successful Login

When enabled, saves the specified cookie with a hashed device identifier value in the client's browser following successful authentication.

amster attribute: `saveDeviceCookieValueOnSuccessfulLogin`

ssoadm attribute: `openam-auth-adaptive-device-cookie-save`

Score

Sets the value to add to the total score if the user fails the Device Registration Cookie Check.

Default: 1

amster attribute: `deviceCookieScore`

ssoadm attribute: `openam-auth-adaptive-device-cookie-score`

Invert Result

When enabled, adds the Score to the total score if the user passes the Device Registration Cookie Check.

amster attribute: `invertDeviceCookieScore`

ssoadm attribute: `openam-auth-adaptive-device-cookie-invert`

Time Since Last Login

The following properties are available under the Time Since Last Login tab:

Time since Last login Check

When enabled, checks whether the client browser request has the specified cookie that holds the encrypted last login time, and check that the last login time is more recent than a maximum number of days you specify.

amster attribute: `timeSinceLastLoginCheckEnabled`

ssoadm attribute: `openam-auth-adaptive-time-since-last-login-check`

Cookie Name

Specifies the name of the cookie holding the encrypted last login time value.

amster attribute: `timeSinceLastLoginCookieName`

ssoadm attribute: `openam-auth-adaptive-time-since-last-login-cookie-name`

Max Time since Last login

Specifies a threshold age of the last login time in days. If the client's last login time is more recent than the number of days specified, then the client successfully passes the check.

amster attribute: `maxTimeSinceLastLogin`

ssoadm attribute: `openam-auth-adaptive-time-since-last-login-value`

Save time of Successful Login

When enabled, saves the specified cookie with the current time encrypted as the last login value in the client's browser following successful authentication.

amster attribute: `saveLastLoginTimeOnSuccessfulLogin`

ssoadm attribute: `openam-auth-adaptive-time-since-last-login-save`

Score

Sets the value to add to the total score if the user fails the Time Since Last Login Check.

Default: 1

amster attribute: `timeSinceLastLoginScore`

ssoadm attribute: `openam-auth-adaptive-time-since-last-login-score`

Invert Result

When enabled, adds the Score to the total score if the user passes the Time Since Last Login Check.

amster attribute: `invertTimeSinceLastLoginScore`

ssoadm attribute: `openam-auth-adaptive-time-since-last-login-invert`

Profile Attribute

The following properties are available under the Profile Attribute tab:

Profile Risk Attribute check

When enabled, checks whether the user profile contains the specified attribute and value.

amster attribute: `profileRiskAttributeCheckEnabled`

ssoadm attribute: `openam-auth-adaptive-risk-attribute-check`

Attribute Name

Specifies the attribute to check on the user profile for the specified value.

amster attribute: `profileRiskAttributeName`

ssoadm attribute: `openam-auth-adaptive-risk-attribute-name`

Attribute Value

Specifies the value to match on the profile attribute. If the attribute is multi-valued, a single match is sufficient to pass the check.

amster attribute: `profileRiskAttributeValue`

ssoadm attribute: `openam-auth-adaptive-risk-attribute-value`

Score

Sets the value to add to the total score if the user fails the Profile Risk Attribute Check.

Default: 1

amster attribute: `profileRiskAttributeScore`

ssoadm attribute: `openam-auth-adaptive-risk-attribute-score`

Invert Result

When enabled, adds the Score to the total score if the user passes the Profile Risk Attribute Check.

amster attribute: `invertProfileRiskAttributeScore`

ssoadm attribute: `openam-auth-adaptive-risk-attribute-invert`

Geo Location

The following properties are available under the Geo Location tab:

Geolocation Country Code Check

When enabled, checks whether the client IP address location matches a country specified in the Valid Country Codes list.

ssoadm attribute: `forgerock-am-auth-adaptive-geo-location-check`

Geolocation Database Location

Path to GeoIP data file used to convert IP addresses to country locations. The geolocation database is not packaged with AM. You can download the GeoIP Country database from MaxMind. Use the binary `.mmdb` file format, rather than `.csv`. You can use the GeoLite Country database for testing.

amster attribute: `geolocationDatabaseLocation`

ssoadm attribute: `openam-auth-adaptive-geo-location-database`

Valid Country Codes

Specifies the list of country codes to match. Use `|` to separate multiple values.

ssoadm attribute: `openam-auth-adaptive-geo-location-values`.

Score

Value to add to the total score if the user fails the Geolocation Country Code Check.

Default: 1

amster attribute: `geolocationScore`

ssoadm attribute: `openam-auth-adaptive-geo-location-score`

Invert Result

When enabled, adds the Score to the total score if the user passes the Geolocation Country Code Check.

amster attribute: `invertGeolocationScore`

ssoadm attribute: `openam-auth-adaptive-geo-location-invert`

Request Header

The following properties are available under the Request Header tab:

Request Header Check

When enabled, checks whether the client browser request has the specified header with the correct value.

amster attribute: `requestHeaderCheckEnabled`

ssoadm attribute: `openam-auth-adaptive-req-header-check`

Request Header Name

Specifies the name of the request header for the Request Header Check.

amster attribute: `requestHeaderName`

ssoadm attribute: `openam-auth-adaptive-req-header-name`

Request Header Value

Specifies the value of the request header for the Request Header Check.

amster attribute: `requestHeaderValue`

ssoadm attribute: `openam-auth-adaptive-req-header-value`

Score

Value to add to the total score if the user fails the Request Header Check.

Default: 1

amster attribute: `requestHeaderScore`

ssoadm attribute: `openam-auth-adaptive-req-header-score`

Invert Result

When enabled, adds the Score to the total score if the user passes the Request Header Check.

amster attribute: `invertRequestHeaderScore`

ssoadm attribute: `openam-auth-adaptive-req-header-invert`

Anonymous Authentication Module Properties

amster service name: `AnonymousModule`

ssoadm service name: `iPlanetAMAuthAnonymousService`

Valid Anonymous Users

Specifies the list of valid anonymous user IDs that can log in without submitting a password.

amster attribute: `validAnonymousUsers`

ssoadm attribute: `iplanet-am-auth-anonymous-users-list`

When user accesses the default module instance login URL, then the module prompts the user to enter a valid anonymous user name.

The default module instance login URL is defined as follows:

```
protocol://hostname:port/deploy_URI/XUI/?module=Anonymous&org=org_name#login
```

Default Anonymous User Name

Specifies the user ID assigned by the module if the Valid Anonymous Users list is empty. The default value is `anonymous`. Note that the anonymous user must be defined in the realm.

amster attribute: `defaultAnonymousUsername`

ssoadm attribute: `iplanet-am-auth-anonymous-default-user-name`

Case Sensitive User IDs

When enabled, determines whether case matters for anonymous user IDs.

amster attribute: `caseSensitiveUsernameMatchingEnabled`

ssoadm attribute: `iplanet-am-auth-anonymous-case-sensitive`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 (default) to any positive integer and is set for each authentication method. The higher number corresponds to a higher level of authentication. If you configured your authentication levels from a 0 to 5 scale, then an authentication level of 5 will require the highest level of authentication.

After a user has authenticated, AM stores the authentication level in the session token. When the user attempts to access a protected resource, the token is presented to the application. The application uses the token's value to determine if the user has the correct authentication level required to access the resource. If the user does not have the required authentication level, the application can prompt the user to authenticate with a higher authentication level.

amster attribute: `authenticationLevel`

ssoadm attribute: `iplanet-am-auth-anonymous-auth-level`

Certificate Authentication Module Properties

amster service name: `CertificateModule`

ssoadm service name: `iPlanetAMAuthCertService`

Match Certificate in LDAP

When enabled, AM searches for a match for the user's certificate in the LDAP directory. If a match is found and not revoked according to a CRL or OCSP validation, then authentication succeeds.

amster attribute: `matchCertificateInLdap`

ssoadm attribute: `iplanet-am-auth-cert-check-cert-in-ldap`

Subject DN Attribute Used to Search LDAP for Certificates

Indicates which attribute and value in the certificate Subject DN is used to find the LDAP entry holding the certificate.

Default: CN

amster attribute: `ldapCertificateAttribute`

ssoadm attribute: `iplanet-am-auth-cert-attr-check-ldap`

Match Certificate to CRL

When enabled, AM checks whether the certificate has been revoked according to a CRL in the LDAP directory.

amster attribute: `matchCertificateToCRL`

ssoadm attribute: `iplanet-am-auth-cert-check-crl`

Issuer DN Attribute Used to Search LDAP for CRLs

Indicates which attribute and value in the certificate Issuer DN is used to find the CRL in the LDAP directory.

Default: CN

If only one attribute is specified, the LDAP search filter used to find the CRL based on the Subject DN of the CA certificate is *`(attr-name=attr-value-in-subject-DN)`*.

For example, if the subject DN of the issuer certificate is `C=US, CN=Some CA, serialNumber=123456`, and the attribute specified is `CN`, then the LDAP search filter used to find the CRL is `(CN=Some CA)`.

In order to distinguish among different CRLs for the same CA issuer, specify multiple attributes separated by commas (`,`) in the same order they occur in the subject DN. When multiple attribute names are provided in a comma-separated list, the LDAP search filter used is *`(cn=attr1=attr1-value-in-subject-DN,attr2=attr2-value-in-subject-DN,...,attrN=attrN-value-in-subject-DN)`*.

For example, if the subject DN of the issuer certificate is `C=US, CN=Some CA, serialNumber=123456`, and the attributes specified are `CN,serialNumber`, then the LDAP search filter used to find the CRL is `(cn=CN=Some CA,serialNumber=123456)`.

amster attribute: `crlMatchingCertificateAttribute`

ssoadm attribute: `iplanet-am-auth-cert-attr-check-crl`

HTTP Parameters for CRL Update

Specifies parameters to be included in any HTTP CRL call to the CA that issued the certificate.

This property supports key pairs of values separated by commas, for example, `param1=value1, param2=value2`.

If the client or CA contains the Issuing Distribution Point Extension, AM uses this information to retrieve the CRL from the distribution point.

amster attribute: `crlHttpParameters`

ssoadm attribute: `iplanet-am-auth-cert-param-get-crl`

Match CA Certificate to CRL

When enabled, AM checks the CRL against the CA certificate to ensure it has not been compromised.

amster attribute: `matchCACertificateToCRL`

ssoadm attribute: `sunAMValidateCACert`

Cache CRLs in memory

(LDAP distribution points only) When enabled, AM caches CRLs.

amster attribute: `cacheCRLsInMemory`

ssoadm attribute: `openam-am-auth-cert-attr-cache-crl`

Update CA CRLs from CRLDistributionPoint

When enabled, AM updates the CRLs stored in the LDAP directory store.

amster attribute: `updateCRLsFromDistributionPoint`

ssoadm attribute: `openam-am-auth-cert-update-crl`

OCSP Validation

When enabled, AM checks the revocation status of certificates using the Online Certificate Status Protocol (OCSP).

You must configure OSCP for AM under Configure > Server Defaults or Deployment > Servers > *Server Name* > Security.

amster attribute: `ocspValidationEnabled`

ssoadm attribute: `iplanet-am-auth-cert-check-ocsp`

LDAP Server Where Certificates are Stored

Identifies the LDAP server that holds users; certificates. The property has the format `ldap_server:port`, for example, `ldap1.example.com:636`. To configure a secure connection, enable the Use SSL/TLS for LDAP Access property.

AM servers can be associated with LDAP servers by writing multiple chains with the format `openam_server|ldapserver:port`, for example, `openam.example.com|ldap1.example.com:636`.

amster attribute: `certificateLdapServers`

ssoadm attribute: `iplanet-am-auth-cert-ldap-provider-url`

LDAP Search Start or Base DN

Valid base DN for the LDAP search, such as `dc=example,dc=com`. To associate AM servers with§ different search base DNs, use the format `openam_server|base_dn`, for example, ` openam.example.com|dc=example,dc=com openam1.test.com|dc=test, dc=com`

amster attribute: `ldapSearchStartDN`

ssoadm attribute: `iplanet-am-auth-cert-start-search-loc`

LDAP Server Authentication User, LDAP Server Authentication Password

If AM stores attributes in the LDAP directory, for example to manage account lockout, or if the LDAP directory requires that AM authenticate in order to read users' attributes, then AM needs the DN and password to authenticate to the LDAP directory.

ssoadm attributes: `iplanet-am-auth-cert-principal-user`, and `iplanet-am-auth-cert-principal-passwd`

Use SSL/TLS for LDAP Access

If you use SSL/TLS for LDAP access, AM must be able to trust the LDAP server certificate.

amster attribute: `sslEnabled`

ssoadm attribute: `iplanet-am-auth-cert-use-ssl`

Certificate Field Used to Access User Profile

If the user profile is in a different entry from the user certificate, then this can be different from subject DN attribute used to find the entry with the certificate. When you select other, provide an attribute name in the Other Certificate Field Used to Access User Profile text box.

amster attribute: `certificateAttributeToProfileMapping`

ssoadm attribute: `iplanet-am-auth-cert-user-profile-mapper`

Valid values: `subject DN`, `subject CN`, `subject UID`, `email address`, `other`, and `none`.

Other Certificate Field Used to Access User Profile

This field is only used if the Certificate Field Used to Access User Profile attribute is set to other. This field allows a custom certificate field to be used as the basis of the user search.

amster attribute: `otherCertificateAttributeToProfileMapping`

ssoadm attribute: `iplanet-am-auth-cert-user-profile-mapper-other`

SubjectAltNameExt Value Type to Access User Profile

Specifies how to look up the user profile:

- Let the property default to `none` to give preference to the Certificate Field Used to Access User Profile or Other Certificate Field Used to Access User Profile attributes when looking up the user profile.

- Select `RFC822Name` if you want AM to look up the user profile from an RFC 822 style name.
- Select `UPN` if you want AM to look up the user profile as the User Principal Name attribute used in Active Directory.

amster attribute: `certificateAttributeProfileMappingExtension`

ssoadm attribute: `iplanet-am-auth-cert-user-profile-mapper-ext`

Trusted Remote Hosts

Defines a list of hosts trusted to send certificates to AM, such as load balancers doing SSL termination.

Valid values are `none`, `any`, and `IP_ADDR`, where `IP_ADDR` is one or more IP addresses of trusted hosts that can send client certificates to AM.

amster attribute: `trustedRemoteHosts`

ssoadm attribute: `iplanet-am-auth-cert-gw-cert-auth-enabled`

HTTP Header Name for Client Certificates

Specifies the name of the HTTP request header containing the PEM-encoded certificate. If Trusted Remote Hosts is set to `any` or specifies the IP address of the trusted host (for example, an SSL-terminated load balancer) that can supply client certificates to AM, the administrator must specify the header name in this attribute.

amster attribute: `clientCertificateHttpHeaderName`

ssoadm attribute: `sunAMHttpParamName`

Use only Certificate from HTTP request header

When enabled, AM always uses the client certificate from the HTTP header rather than the certificate the servlet container receives during the SSL handshake.

Default: false

ssoadm attribute: `iplanet-am-auth-cert-gw-cert-preferred`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

amster attribute: `authenticationLevel`

ssoadm attribute: `iplanet-am-auth-cert-auth-level`

Data Store Authentication Module Properties

amster service name: `DataStoreModule`

ssoadm service name: `sunAMAuthDataStoreService`

Authentication Level

> Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.
>
> **amster** attribute: `authenticationLevel`
>
> **ssoadm** attribute: `sunAMAuthDataStoreAuthLevel`

Device ID (Match) Authentication Module Properties

amster service name: `DeviceIdMatchModule`

ssoadm service name: `iPlanetAMAuthDeviceIdMatchService`

Client-Side Script Enabled

> Enable Device ID (Match) to send JavaScript in an authentication page to the device to collect data about the device by a self-submitting form.
>
> **amster** attribute: `clientScriptEnabled`
>
> **ssoadm** attribute: `iplanet-am-auth-scripted-client-script-enabled`

Client-Side Script, Server-Side Script

> Specify the client-side and server-side Javascript scripts to use with the Device Id (Match) module.
>
> To view and modify the contents of the scripts, navigate to Realms > *Realm Name* > Scripts and select the name of the script.
>
> If you change the client-side script, you must make a corresponding change in the server-side script to account for the specific addition or removal of an element.
>
> **ssoadm** attribute: `iplanet-am-auth-scripted-client-script` and `iplanet-am-auth-scripted-server-script`

Authentication Level

> Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

amster attribute: `authenticationLevel`

ssoadm attribute: `iplanet-am-auth-scripted-auth-level`

Device ID (Save) Authentication Module Properties

amster service name: `DeviceIdSaveModule`

ssoadm service name: `iPlanetAMAuthDeviceIdSaveService`

Automatically store new profiles

When enabled, AM assumes user consent to store new profiles. After successful HOTP confirmation, AM stores the new profile automatically.

amster attribute: `autoStoreProfiles`

ssoadm attribute: `iplanet-am-auth-device-id-save-auto-store-profile`

Maximum stored profile quantity

Sets the maximum number of stored profiles on the user's record.

amster attribute: `maxProfilesAllowed`

ssoadm attribute: `iplanet-am-auth-device-id-save-max-profiles-allowed`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

amster attribute: `authenticationLevel`

ssoadm attribute: `iplanet-am-auth-device-id-save-auth-level`

Federation Authentication Module Properties

amster service name: `FederationModule`

ssoadm service name: `sunAMAuthFederationService`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

amster attribute: `authenticationLevel`

ssoadm attribute: `sunAMAuthFederationAuthLevel`

Amster Authentication Module Properties

amster service name: `AmsterModule`

ssoadm service name: `iPlanetAMAuthAmsterService`

Authorized Keys

Specifies the location of the `authorized_keys` file that contains the private and public keys used to validate remote **amster** client connections.

The default location for the `authorized_keys` file is the `/path/to/openam/` path. Its content is similar to an OpenSSH `authorized_keys` file.

amster attribute: `forgerock-am-auth-amster-authorized-keys`

Enabled

When enabled, allows **amster** clients to authenticate using PKI. When disabled, allows **amster** clients to authenticate using interactive login only.

amster attribute: `forgerock-am-auth-amster-enabled`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

amster attribute: `forgerock-am-auth-amster-auth-level`

ForgeRock Authenticator (OATH) Authentication Module Properties

amster service name: `AuthenticatorOathModule`

ssoadm service name: `iPlanetAMAuthAuthenticatorOATHService`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

ssoadm attribute: `iplanet-am-auth-fr-oath-auth-level`

One-Time Password Length

Sets the length of the OTP to six digits or longer. The default value is six.

amster attribute: `passwordLength`

ssoadm attribute: `iplanet-am-auth-fr-oath-password-length`

Minimum Secret Key Length

The minimum number of hexadecimal characters allowed for the secret key.

amster attribute: `minimumSecretKeyLength`

ssoadm attribute: `iplanet-am-auth-fr-oath-min-secret-key-length`

OATH Algorithm to Use

Select whether to use HOTP or TOTP. You can create an authentication chain to allow for a greater variety of devices. The default value is HOTP.

amster attribute: `oathAlgorithm`

ssoadm attribute: `iplanet-am-auth-fr-oath-algorithm`

HOTP Window Size

The window that the OTP device and the server counter can be out of sync. For example, if the window size is 100 and the server's last successful login was at counter value 2, then the server will accept an OTP from device counter 3 to 102. The default value is 100.

amster attribute: `hotpWindowSize`

ssoadm attribute: `iplanet-am-auth-fr-oath-hotp-window-size`

Add Checksum Digit

Adds a checksum digit at the end of the HOTP password to verify the OTP was generated correctly. This is in addition to the actual password length. Set this only if your device supports it. The default value is No.

amster attribute: `addChecksumToOtpEnabled`

ssoadm attribute: `iplanet-am-auth-fr-oath-add-checksum`

Truncation Offset

Advanced feature that is device-specific. Let this value default unless you know your device uses a truncation offset. The default value is -1.

amster attribute: `truncationOffset`

ssoadm attribute: `iplanet-am-auth-fr-oath-truncation-offset`

TOTP Time Step Interval

The time interval for which an OTP is valid. For example, if the time step interval is 30 seconds, a new OTP will be generated every 30 seconds, and an OTP will be valid for 30 seconds. The default value is 30 seconds.

amster attribute: `totpTimeStepInterval`

ssoadm attribute: `iplanet-am-auth-fr-oath-size-of-time-step`

TOTP Time Steps

The number of time step intervals that the system and the device can be off before password resynchronization is required. For example, if the number of TOTP time steps is 2 and the TOTP time step interval is 30 seconds, the server will allow an 89 second clock skew between the client and the server—two 30 second steps plus 29 seconds for the interval in which the OTP arrived. The default value is 2.

amster attribute: `totpTimeStepsInWindow`

ssoadm attribute: `iplanet-am-auth-fr-oath-steps-in-window`

One Time Password Max Retry

The number of times entry of the OTP may be attempted. Minimum is 1, maximum is 10.

Default: 3

amster attribute: `oathOtpMaxRetry`

ssoadm attribute: `forgerock-oath-max-retry`

Maximum Allowed Clock Drift

The maximum acceptable clock skew before authentication fails. When this value is exceeded, the user must re-register the device.

amster attribute: `totpMaximumClockDrift`

ssoadm attribute: `openam-auth-fr-oath-maximum-clock-drift`

Name of the Issuer

A value that appears as an identifier on the user's device. Common choices are a company name, a web site, or an AM realm.

amster attribute: `oathIssuerName`

ssoadm attribute: `openam-auth-fr-oath-issuer-name`

ForgeRock Authenticator (Push) Authentication Module Properties

amster service name: `AuthenticatorPushModule`

ssoadm service name: `iPlanetAMAuthAuthenticatorPushService`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

amster attribute: `authenticationLevel`

ssoadm attribute: `forgerock-am-auth-authenticatorpush-auth-level`

Return Message Timeout (ms)

The period of time (in milliseconds) within which a push notification should be replied to.

Default: `120000`

amster attribute: `timeoutInMilliSecconds`

ssoadm attribute: `forgerock-am-auth-push-message-response-timeout`

Login Message

Text content of the push message, which is used for the notification displayed on the registered device. The following variables can be used in the message:

`{{user}}`

Replaced with the username value of the account registered in the ForgeRock Authenticator app, for example *Demo*.

`{{issuer}}`

Replaced with the issuer value of the account registered in the ForgeRock Authenticator app, for example *ForgeRock*.

Default: `Login attempt from {{user}} at {{issuer}}`

amster attribute: `pushMessage`

ssoadm attribute: `forgerock-am-auth-push-message`

ForgeRock Authenticator (Push) Registration Authentication Module Properties

amster service name: `AuthenticatorPushRegistrationModule`

ssoadm service name: `iPlanetAMAuthAuthenticatorPushRegistrationService`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

amster attribute: `authenticationLevel`

ssoadm attribute: `forgerock-am-auth-push-reg-auth-level`

Issuer Name

A value that appears as an identifier on the user's device. Common choices are a company name, a web site, or an AM realm.

amster attribute: `issuer`

ssoadm attribute: `forgerock-am-auth-push-reg-issuer`

Registration Response Timeout (ms)

The period of time (in milliseconds) to wait for a response to the registration QR code. If no response is received during this time the QR code times out and the registration process fails.

Default: `120000`

amster attribute: `timeoutInMilliSecconds`

ssoadm attribute: `forgerock-am-auth-push-message-registration-response-timeout`

Background Color

The background color in hex notation to display behind the issuer's logo within the ForgeRock Authenticator app.

Default: `#519387`

amster attribute: `bgcolour`

ssoadm attribute: `forgerock-am-auth-hex-bgcolour`

Image URL

The location of an image to download and display as the issuer's logo within the ForgeRock Authenticator app.

amster attribute: `imgUrl`

ssoadm attribute: `forgerock-am-auth-img-url`

App Store App URL

URL of the app to download on the App Store.

Default: `https://itunes.apple.com/app/forgerock-authenticator /id1038442926` (the ForgeRock Authenticator app)

amster attribute: `appleLink`

ssoadm attribute: `forgerock-am-auth-apple-link`

Google Play URL

URL of the app to download on Google Play.

Default: `https://play.google.com/store/apps/details?id=com.forgerock.authenticator` (the ForgeRock Authenticator app)

amster attribute: `googleLink`

ssoadm attribute: `forgerock-am-auth-google-link`

HOTP Authentication Module Properties

amster service name: `HotpModule`

ssoadm service name: `sunAMAuthHOTPService`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

amster attribute: `authenticationLevel`

ssoadm attribute: `sunAMAuthHOTPAuthLevel`

SMS Gateway Implementation Class

Specifies the class the HOTP module uses to send SMS or email messages. Specify a class that implements the `com.sun.identity.authentication.modules.hotp.SMSGateway` interface to customize the SMS gateway implementation.

amster attribute: `smsGatewayClass`

ssoadm attribute: `sunAMAuthHOTPSMSGatewayImplClassName`

Mail Server Host Name

Specifies the hostname of the mail server supporting SMTP for electronic mail.

amster attribute: `smtpHostname`

ssoadm attribute: `sunAMAuthHOTPSMTPHostName`

Mail Server Host Port

Specifies the outgoing mail server port. The default port is 25, 465 (when connecting over SSL), or 587 (for StartTLS).

amster attribute: `smtpHostPort`

ssoadm attribute: `sunAMAuthHOTPSMTPHostPort`

Mail Server Authentication Username

Specifies the username for AM to connect to the mail server.

amster attribute: `smtpUsername`

ssoadm attribute: `sunAMAuthHOTPSMTPUserName`

Mail Server Authentication Password

Specifies the password for AM to connect to the mail server.

amster attribute: `smtpUserPassword`

ssoadm attribute: `sunAMAuthHOTPSMTPUserPassword`

Mail Server Secure Connection

Specifies whether to connect to the mail server securely. If enabled, AM must be able to trust the server certificate.

The possible values for this property are:

```
SSL
Non SSL
Start TLS
```

amster attribute: `smtpSslEnabled`

ssoadm attribute: `sunAMAuthHOTPSMTPSSLEnabled`

Email From Address

Specifies the `From:` address when sending a one-time password by mail.

amster attribute: `smtpFromAddress`

ssoadm attribute: `sunAMAuthHOTPSMTPFromAddress`

One-Time Password Validity Length (in minutes)

Specifies the amount of time, in minutes, the one-time passwords are valid after they are generated. The default is `5` minutes.

amster attribute: `otpValidityDuration`

ssoadm attribute: `sunAMAuthHOTPPasswordValidityDuration`

One-Time Password Length

Sets the length of one-time passwords.

amster attribute: `otpLength`

ssoadm attribute: `sunAMAuthHOTPPasswordLength`

Valid values: `6` and `8`.

One Time Password Max Retry

The number of times entry of the OTP may be attempted. Minimum is 1, maximum is 10.

Default: 3

amster attribute: `oathOtpMaxRetry`

ssoadm attribute: `forgerock-oath-max-retry`

One-Time Password Delivery

Specifies whether to send the one-time password by SMS, by mail, or both.

amster attribute: `otpDeliveryMethod`

ssoadm attribute: `sunAMAuthHOTPasswordDelivery`

Valid values: `SMS`, `E-mail`, and `SMS and E-mail`.

Mobile Phone Number Attribute Name

Provides the attribute name used for the text message. The default value is `telephoneNumber`.

amster attribute: `userProfileTelephoneAttribute`

ssoadm attribute: `openamTelephoneAttribute`

Mobile Carrier Attribute Name

Specifies a user profile attribute that contains a mobile carrier domain for sending SMS messages.

The uncustomized AM user profile does not have an attribute for the mobile carrier domain. You can:

- Customize the AM user profile by adding a new attribute to it. Then you can populate the new attribute with users' SMS messaging domains.

  All mobile carriers and bulk SMS messaging services have associated SMS messaging domains. For example, Verizon uses `vtext.com`, T-Mobile uses `tmomail.net`, and the TextMagic service uses `textmagic.com`. If you plan to send text messages internationally, determine whether the messaging service requires a country code.

- Leave the value for Mobile Carrier Attribute Name blank, and let AM default to sending SMS messages using `txt.att.net` for all users.

amster attribute: `mobileCarrierAttribute`

ssoadm attribute: `openamSMSCarrierAttribute`

Email Attribute Name

Provides the attribute name used to email the OTP. The default value is `mail` (email).

amster attribute: `userProfileEmailAttribute`

ssoadm attribute: `openamEmailAttribute`

Auto Send OTP Code

When enabled, configures the HOTP module to automatically generate an email or text message when users begin the login process.

ssoadm attribute: `sunAMAuthHOTPAutoClicking`

HTTP Basic Authentication Module Properties

amster service name: `HttpBasicModule`

ssoadm service name: `iPlanetAMAuthHTTPBasicService`

Backend Module Name

Specifies the module that checks the user credentials. The credentials are then supplied to either a data store or other identity repository module for authentication.

amster attribute: `backendModuleName`

ssoadm attribute: `iplanet-am-auth-http-basic-module-configured`

Valid values: `LDAP` and `DataStore`.

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

amster attribute: `authenticationLevel`

ssoadm attribute: `iplanet-am-auth-httpbasic-auth-level`

JDBC Authentication Module Properties

amster service name: `JdbcModule`

ssoadm service name: `sunAMAuthJDBCService`

Connection Type

Determines how the module obtains the connection to the database.

amster attribute: `connectionType`

ssoadm attribute: `sunAMAuthJDBCConnectionType`

Valid values: `JNDI` and `JDBC`.

Connection Pool JNDI Name

Specifies the URL of the connection pool for JNDI connections. Refer to your web container's documentation for instructions on setting up the connection pool.

amster attribute: `connectionPoolJndiName`

ssoadm attribute: `sunAMAuthJDBCJndiName`

JDBC Driver

Specifies the JDBC driver to use for JDBC connections.

Install a suitable Oracle or MySQL driver in the container where AM is installed, for example in the `/path/to/tomcat/webapps/openam/WEB-INF/lib` path. You can add it to the AM `.war` file when you deploy AM.

amster attribute: `jdbcDriver`

ssoadm attribute: `sunAMAuthJDBCDriver`

JDBC URL

Specifies the URL to connect to the database when using a JDBC connection.

amster attribute: `jdbcUrl`

ssoadm attribute: `sunAMAuthJDBCUrl`

Database Username, Database Password

Specifies the user name and password used to authenticate to the database when using a JDBC connection.

ssoadm attribute: `sunAMAuthJDBCDbuser` and `sunAMAuthJDBCDbpassword`

Password Column Name

Specifies the database column name where passwords are stored.

amster attribute: `passwordColumn`

ssoadm attribute: `sunAMAuthJDBCPasswordColumn`

Prepared Statement

Specifies the SQL query to return the password corresponding to the user to authenticate.

amster attribute: `passwordStatement`

ssoadm attribute: `sunAMAuthJDBCStatement`

Class to Transform Password Syntax

Specifies the class that transforms the password retrieved to the same format as provided by the user.

The default class expects the password in cleartext. Custom classes must implement the `JDBCPasswordSyntaxTransform` interface.

amster attribute: `passwordTransformClass`

ssoadm attribute: `sunAMAuthJDBCPasswordSyntaxTransformPlugin`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

amster attribute: `authenticationLevel`

ssoadm attribute: `sunAMAuthJDBCAuthLevel`

> **Note**
>
> AM provides two properties, `iplanet-am-admin-console-invalid-chars` and `iplanet-am-auth-ldap-invalid-chars`, that store LDAP-related special characters that are not allowed in username searches.
>
> When using JDBC databases, consider adding the '%' wildcard character to the `iplanet-am-admin-console-invalid-chars` and `iplanet-am-auth-ldap-invalid-chars` properties. By default, the '%' character is not included in the properties.

LDAP Authentication Module Properties

amster service name: `LdapModule`

ssoadm service name: `iPlanetAMAuthLDAPService`

Primary LDAP Server
Secondary LDAP Server

Directory servers generally use built-in data replication for high availability. Thus, a directory service likely consists of a pool of replicas to which AM can connect to retrieve and update directory data. You set up primary and secondary servers in case a replica is down due to maintenance or to a problem with a particular server.

Set one or more primary and optionally, one or more secondary directory server for each AM server. For the current AM server, specify each directory server as a `*host*:*port*` combination. For other AM servers in the deployment, you can specify each directory server as `*server-name*|*host*:*port*`, where *server-name* is the FQDN portion of the AM server from the list under Deployment > Servers, and *host*:*port* identifies the directory server.

For example, if the *server-name* that is listed is `https://openam.example.com:8443/openam`, and the directory server is accessible at `opendj.example.com:1389`, you would enter `openam.example.com|opendj.example.com:1389`.

When authenticating users from a directory server that is remote to AM, set the primary server values, and optionally the secondary server values. Primary servers have priority over secondary servers.

ssoadm attributes are: primary is `iplanet-am-auth-ldap-server`; secondary is `iplanet-am-auth-ldap-server2`.

Both properties take more than one value; thus, allowing more than one primary or secondary remote server, respectively. Assuming a multi-data center environment, AM determines priority within the primary and secondary remote servers, respectively, as follows:

- Every LDAP server that is mapped to the current AM instance has highest priority.

  For example, if you are connected to `openam1.example.com` and `ldap1.example.com` is mapped to that AM instance, then AM uses `ldap1.example.com`.

- Every LDAP server that was not specifically mapped to a given AM instance has the next highest priority.

  For example, if you have another LDAP server, `ldap2.example.com`, that is not connected to a specific AM server and if `ldap1.example.com` is unavailable, AM connects to the next highest priority LDAP server, `ldap2.example.com`.

- LDAP servers that are mapped to different AM instances have the lowest priority.

  For example, if `ldap3.example.com` is connected to `openam3.example.com` and `ldap1.example.com` and `ldap2.example.com` are unavailable, then `openam1.example.com` connects to `ldap3.example.com`.

If you want use SSL or StartTLS to initiate a secure connection to a data store, then scroll down to enable SSL/TLS Access to LDAP Server. Make sure that AM can trust the server's certificates when using this option.

ssoadm attributes: `openam-auth-ldap-connection-mode`

Possible values: `LDAP`, `LDAPS`, and `StartTLS`

DN to Start User Search

LDAP data is organized hierarchically, a bit like a file system on Windows or UNIX. More specific DNs likely result in better search performance. When configuring the module for a particular part of the organization, you can perhaps start searches from a specific organizational unit, such as `ou=sales,dc=example,dc=com`.

If multiple entries exist with identical search attribute values, make this value specific enough to return only one entry.

ssoadm attribute: `iplanet-am-auth-ldap-base-dn`

Bind User DN, Bind User Password

If AM stores attributes in the directory, for example to manage account lockout, or if the directory requires that AM authenticate in order to read users' attributes, then AM needs the DN and password to authenticate to the directory.

The default is `cn=Directory Manager`. Make sure that password is correct before you log out. If it is incorrect, you will be locked out. If this should occur, you can login with the superuser DN, which by default is `uid=amAdmin,ou=People,`*`AM-deploy-base`*, where *AM-deploy-base* is the value you set during AM configuration.

ssoadm attributes: `iplanet-am-auth-ldap-bind-dn`, `iplanet-am-auth-ldap-bind-passwd`

Attribute Used to Retrieve User Profile
Attributes Used to Search for a User to be Authenticated
User Search Filter
Search Scope

LDAP searches for user entries return entries with attribute values matching the filter you provide. For example, if you search under `ou=people,dc=example,dc=com` with a filter `"(mail=bjensen@example.com)"`, then the directory returns the entry that has `mail=bjensen@example.com`. In this example the attribute used to search for a user is `mail`. Multiple attribute values mean the user can authenticate with any one of the values. For example, if you have both `uid` and `mail`, then Barbara Jensen can authenticate with either `bjensen` or `bjensen@example.com`.

Should you require a more complex filter for performance, you add that to the User Search Filter text box. For example, if you search on `mail` and add User Search Filter `(objectClass=inetOrgPerson)`, then AM uses the resulting search filter `(&(mail=`*`address`*`)(objectClass=inetOrgPerson))`, where *address* is the mail address provided by the user.

Scope OBJECT means search only the entry specified as the DN to Start User Search, whereas ONELEVEL means search only the entries that are directly children of that object. SUBTREE means search the entry specified and every entry under it.

ssoadm attributes: `iplanet-am-auth-ldap-user-naming-attribute`, `iplanet-am-auth-ldap-user-search-attributes`, `iplanet-am-auth-ldap-search-filter`, and `iplanet-am-auth-ldap-search-scope`

LDAP Connection Mode

If you want use SSL or StartTLS to initiate a secure connection to a data store, AM must be able to trust LDAP certificates, either because the certificates were signed by a CA whose certificate is already included in the trust store used by the container where AM runs, or because you imported the certificates into the trust store.

ssoadm attribute: `openam-auth-ldap-connection-mode`

Possible values: `LDAP`, `LDAPS`, and `StartTLS`

Return User DN to DataStore

When enabled, and AM uses the directory service as the user store, the module returns the DN, rather than the User ID. From the DN value, AM uses the RDN to search for the user profile. For example, if a returned DN value is `uid=demo,ou=people,dc=openam,dc=example,dc=org`, AM uses `uid=demo` to search the data store.

amster attribute: `returnUserDN`

ssoadm attribute: `iplanet-am-auth-ldap-return-user-dn`

User Creation Attributes

This list lets you map (external) attribute names from the LDAP directory server to (internal) attribute names used by AM.

amster attribute: `profileAttributeMappings`

ssoadm attribute: `iplanet-am-ldap-user-creation-attr-list`

Minimum Password Length

Specifies the minimum acceptable password length.

amster attribute: `minimumPasswordLength`

ssoadm attribute: `iplanet-am-auth-ldap-min-password-length`

LDAP Behera Password Policy Support

When enabled, support interoperability with servers that implement the Internet-Draft, Password Policy for LDAP Directories.

Support for this Internet-Draft is limited to the LDAP authentication module. Other components of AM, such as the password change functionality in the `/idm/EndUser` page, do not support the Internet-Draft. In general, outside of the LDAP authentication module, AM binds to the directory server as an administrator, such as Directory Manager. When AM binds to the directory server as an administrator rather than as an end user, many features of the Internet-Draft password policies do not apply.

amster attribute: `beheraPasswordPolicySupportEnabled`

ssoadm attribute: `iplanet-am-auth-ldap-behera-password-policy-enabled`

Trust All Server Certificates

When enabled, blindly trust server certificates, including self-signed test certificates.

amster attribute: `trustAllServerCertificates`

ssoadm attribute: `iplanet-am-auth-ldap-ssl-trust-all`

LDAP Connection Heartbeat Interval

Specifies how often AM should send a heartbeat request to the directory server to ensure that the connection does not remain idle. Some network administrators configure firewalls and load balancers to drop connections that are idle for too long. You can turn this off by setting the value to 0 or to a negative number. To set the units for the interval use LDAP Connection Heartbeat Time Unit.

Default: 1

amster attribute: `connectionHeartbeatInterval`

ssoadm attribute: `openam-auth-ldap-heartbeat-interval`

LDAP Connection Heartbeat Time Unit

Specifies the time unit corresponding to LDAP Connection Heartbeat Interval.

Default: minute

amster attribute: `connectionHeartbeatTimeUnit`

ssoadm attribute: `openam-auth-ldap-heartbeat-timeunit`

LDAP operations timeout

Defines the timeout in milliseconds that AM should wait for a response from the directory server.

Default: 0 (means no timeout)

amster attribute: `operationTimeout`

ssoadm attribute: `openam-auth-ldap-operation-timeout`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

amster attribute: `authenticationLevel`

ssoadm attribute: `iplanet-am-auth-ldap-auth-level`

Legacy OAuth 2.0/OpenID Connect Authentication Module Properties

Important

Usage of this authentication module is deprecated. Equivalent functionality is provided by the following authentication modules:

- Social Authentication Module Properties - OAuth 2.0
- Social Authentication Module Properties - OpenID Connect 1.0

The Legacy OAuth 2.0/OpenID Connect Authentication Module will only be available in AM when upgrading from a previous version that was making use of the module in a chain. It is not available in new, clean installations since AM 5.5.

The default settings are for Facebook.

amster service name: `OAuth2Module`

ssoadm service name: `sunAMAuthOAuthService`

Client id

Specifies the OAuth 2.0 `client_id` parameter as described in section 2.2 of RFC 6749.

amster attribute: `clientId`

ssoadm attribute: `iplanet-am-auth-oauth-client-id`

Client Secret

Specifies the OAuth 2.0 `client_secret` parameter as described in section 2.3 of RFC 6749.

amster attribute: `clientSecret`

ssoadm attribute: `iplanet-am-auth-oauth-client-secret`

Authentication Endpoint URL

Specifies the URL to the endpoint handling OAuth 2.0 authentication as described in section 3.1 of RFC 6749.

Default:`https://www.facebook.com/dialog/oauth`.

amster attribute: `authenticationEndpointUrl`

ssoadm attribute: `iplanet-am-auth-oauth-auth-service`

Access Token Endpoint URL

Specifies the URL to the endpoint handling access tokens as described in section 3.2 of RFC 6749.

Default:`https://graph.facebook.com/oauth/access_token`.

amster attribute: `accessTokenEndpointUrl`

ssoadm attribute: `iplanet-am-auth-oauth-token-service`

User Profile Service URL

Specifies the user profile URL that returns profile information in JSON format.

Default:`https://graph.facebook.com/me`.

amster attribute: `userProfileServiceUrl`

ssoadm attribute: `iplanet-am-auth-oauth-user-profile-service`

Scope

Specifies a space-delimited list of user profile attributes that the client application requires, according to *The OAuth 2.0 Authorization Framework*. The list depends on the permissions that the resource owner, such as the end user, grants to the client application.

Some authorization servers use non-standard separators for scopes. Facebook, for example, takes a comma-separated list.

Default: `email,read_stream` (Facebook example)

amster attribute: `scope`

ssoadm attribute: `iplanet-am-auth-oauth-scope`

OAuth2 Access Token Profile Service Parameter name

Specifies the name of the parameter that contains the access token value when accessing the profile service.

Default: `access_token`.

amster attribute: `accessTokenParameterName`

ssoadm attribute: `iplanet-am-auth-oauth-user-profile-param`

Proxy URL

Sets the URL to the `/oauth2c/OAuthProxy.jsp` file, which provides AM with GET to POST proxying capabilities. Change this URL only if an external server performs the GET to POST proxying.

Default: `@SERVER_PROTO@://@SERVER_HOST@:@SERVER_PORT@/@SERVER_URI@/oauth2c/OAuthProxy.jsp`.

amster attribute: `ssoProxyUrl`

ssoadm attribute: `iplanet-am-auth-oauth-sso-proxy-url`

Account Provider

Specifies the name of the class that implements the account provider.

Default: `org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider`

amster attribute: `accountProviderClass`

ssoadm attribute: `org-forgerock-auth-oauth-account-provider`

Account Mapper

Specifies the name of the class that implements the attribute mapping for the account search.

Default: Depends on how the module is created:

- If the OAuth 2.0 authentication module is created from the AM console authentication tab of a realm, the default is: `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper`.

- If the OAuth 2.0 authentication module is created from the AM console Facebook authentication wizard, the default is: `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper|*|facebook-`.
- If the OAuth 2.0 authentication module is created from the AM console Google authentication wizard, the default is: `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper|*|Google-`.

amster attribute: `accountMapperClass`

ssoadm attribute: `org-forgerock-auth-oauth-account-mapper`

Account Mapper Configuration

Specifies the attribute configuration used to map the account of the user authenticated in the OAuth 2.0 provider to the local data store in AM. Valid values are in the form *`provider-attr=local-attr`*.

> **Tip**
>
> When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings, by using dot notation.
>
> For example, given a JSON payload of:
>
> ```
> {
>   "sub" : "12345",
>   "name" : {
>     "first_name" : "Demo",
>     "last_name" : "User"
>   }
> }
> ```
>
> You can create a mapper such as:
>
> `name.first_name=cn`

Default: `email=mail` and `id=facebook-id`.

amster attribute: `accountMapperConfiguration`

ssoadm attribute: `org-forgerock-auth-oauth-account-mapper-configuration`

Attribute Mapper

Specifies the list of fully qualified class names for implementations that map attributes from the OAuth 2.0 authorization server or OpenID Connect provider to AM profile attributes.

Default: `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper`

Provided implementations are:

`org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper` (can only be used when using the `openid` scope)

Tip

You can provide string constructor parameters by appending pipe (`|`) separated values.

For example, the `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper` class can take two constructor parameters: a comma-separated list of attributes and a prefix to apply to their values. Specify these as follows:

`org.forgerock.openam.authentication.modules.oidc.JsonAttributeMapper`

amster attribute: `attributeMappingClasses`

ssoadm attribute: `org-forgerock-auth-oauth-attribute-mapper`

Attribute Mapper Configuration

Map of OAuth 2.0 provider user account attributes to local user profile attributes, with values in the form *`provider-attr=local-attr`*.

Tip

When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings, by using dot notation.

For example, given a JSON payload of:

```
{
  "sub" : "12345",
  "name" : {
    "first_name" : "Demo",
    "last_name" : "User"
  }
}
```

You can create a mapper such as:

`name.first_name=cn`

Default: `first_name=givenname`, `last_name=sn`, `name=cn`, `email=mail`, `id=facebook-id`, `first_name=facebook-fname`, `last_name=facebook-lname`, `email=facebook-email`.

amster attribute: `attributeMapperConfiguration`

ssoadm attribute: `org-forgerock-auth-oauth-attribute-mapper-configuration`

Save attributes in the session

When enabled, saves the attributes in the Attribute Mapper Configuration field to the AM session.

amster attribute: `saveAttributesInSession`

ssoadm attribute: `org-forgerock-auth-oauth-save-attributes-to-session-flag`

Email attribute in OAuth2 Response

Specifies the attribute identifying the authenticated user's email address in the response from the profile service in the OAuth 2.0 provider. This setting is used to send an email message with an activation code for accounts created dynamically.

amster attribute: `oauth2EmailAttribute`

ssoadm attribute: `org-forgerock-auth-oauth-mail-attribute`

Create account if it does not exist

When enabled, AM creates an account for the user if the user profile does not exist. If the Prompt for password setting and activation code attribute is enabled, AM prompts the user for a password and activation code before creating the account.

When the OAuth 2.0/OpenID Connect client is configured to create new accounts, the SMTP settings must also be valid. As part of account creation, the OAuth 2.0/OpenID Connect client authentication module sends the resource owner an email with an account activation code. To send the mail, AM uses the SMTP settings you provide here in the OAuth 2.0/OpenID Connect client configuration.

When disabled, a user without a profile may still log into AM if the Ignore Profile attribute is set in the authentication service of the realm, or if the account is mapped to an anonymous account.

amster attribute: `createAccount`

ssoadm attribute: `org-forgerock-auth-oauth-createaccount-flag`

Prompt for password setting and activation code

When enabled, the user must set a password before AM creates an account dynamically. An activation code is also sent to the user's email address. Both the password and the code are required before the account is created.

amster attribute: `promptForPassword`

ssoadm attribute: `org-forgerock-auth-oauth-prompt-password-flag`

Map to anonymous user

When enabled, maps the OAuth 2.0 authenticated user to the specified anonymous user. If the Create account if it does not exist property is enabled, AM creates an account for the authenticated user instead of mapping the account to the anonymous user.

amster attribute: `mapToAnonymousUser`

ssoadm attribute: `org-forgerock-auth-oauth-map-to-anonymous-flag`

Anonymous User

Specifies an anonymous user that exists in the current realm. The Map to anonymous user property maps authorized users without a profile to this anonyomus user, if enabled.

Default: `anonymous`.

amster attribute: `anonymousUserName`

ssoadm attribute: `org-forgerock-auth-oauth-anonymous-user`

OAuth 2.0 Provider logout service

Specifies the optional URL of the OAuth 2.0 provider's logout service, if required.

amster attribute: `oauth2LogoutServiceUrl`

ssoadm attribute: `org-forgerock-auth-oauth-logout-service-url`

Logout options

Specifies whether not to log the user out without prompting from the OAuth 2.0 provider on logout, to log the user out without prompting, or to prompt the user regarding whether to log out from the OAuth 2.0 provider.

Valid values are:

- `prompt`, to ask the user whether or not to log out from the OAuth 2.0 provider.
- `logout`, to log the user out of the OAuth 2.0 provider without prompting.
- `donotlogout`, to keep the user logged in to the OAuth 2.0 provider. There is no prompt to the user.

Default: `prompt`.

amster attribute: `logoutBehaviour`

ssoadm attribute: `org-forgerock-auth-oauth-logout-behaviour`

Mail Server Gateway implementation class

Specifies the class used by the module to send email. A custom subclass of `org.forgerock.openam.authentication.modules.oauth2.EmailGateway` class can be provided.

Default: `org.forgerock.openam.authentication.modules.oauth2.DefaultEmailGatewayImpl`

amster attribute: `mailGatewayClass`

ssoadm attribute: `org-forgerock-auth-oauth-email-gwy-impl`

SMTP host

Specifies the host name of the mail server.

Default: `localhost`.

amster attribute: `smtpHostName`

ssoadm attribute: `org-forgerock-auth-oauth-smtp-hostname`

SMTP port

Specifies the SMTP port number for the mail server.

Default: `25`.

amster attribute: `smtpHostPort`

ssoadm attribute: `org-forgerock-auth-oauth-smtp-port`

SMTP User Name, SMTP User Password

Specifies the username and password AM uses to authenticate to the mail server.

ssoadm attribute: `org-forgerock-auth-oauth-smtp-username` and `org-forgerock-auth-oauth-smtp-password`.

SMTP SSL Enabled

When enabled, connects to the mail server over SSL. AM must be able to trust the SMTP server certificate.

amster attribute: `smtpSslEnabled`

ssoadm attribute: `org-forgerock-auth-oauth-smtp-ssl_enabled`

SMTP From address

Specifies the address of the email sender, such as `no-reply@example.com`.

Default: `info@forgerock.com`.

amster attribute: `smtpFromAddress`

ssoadm attribute: `org-forgerock-auth-oauth-smtp-email-from`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

Default: 0.

amster attribute: `authenticationLevel`

ssoadm attribute: `iplanet-am-auth-oauth-auth-level`

OpenID Connect validation configuration type

Validates the ID token from the OpenID Connect provider. The module needs either a URL to get the public keys for the provider or the symmetric key for an ID token signed with a HMAC-based algorithm.

By default, the configuration type is `.well-known/openid-configuration_url`. This means the module should retrieve the keys based on information in the OpenID Connect provider configuration document.

You can instead configure the authentication module to validate the ID token signature with the client secret key you provide, or to validate the ID token with the keys retrieved from the URL to the OpenID Connect provider's JSON web key set.

`/oauth2/realms/root/.well-known/openid-configuration_url` (Default)

Retrieve the provider keys based on the information provided in the OpenID Connect Provider Configuration Document.

Specify the URL to the document as the discovery URL.

`client_secret`

Use the client secret that you specify as the key to validate the ID token signature according to the HMAC by using the client secret to the decrypt the hash, and then checking that the hash matches the hash of the ID token JWT.

`jwk_url`

Retrieve the provider's JSON web key set as the URL that you specify.

amster attribute: `cryptoContextType`

ssoadm attribute: `openam-auth-openidconnect-crypto-context-type`

OpenID Connect validation configuration value

Edit this field depending on the Configuration type you specified in the OpenId Connect validation configuration type field.

amster attribute: `cryptoContextValue`

ssoadm attribute: `openam-auth-openidconnect-crypto-context-value`

Token Issuer

Required when the `openid` scope is included. Value must match the `iss` field in the issued ID token. For example, `accounts.google.com`.

The issuer value MUST be provided when OAuth 2.0 Mix-Up Mitigation is enabled. For more information, see "OAuth 2.0 Mix-Up Mitigation".

amster attribute: `idTokenIssuer`

ssoadm attribute: `openam-auth-openidconnect-issuer-name`

> **Note**
>
> Old uses of `DefaultAccountMapper` are automatically upgraded to the equivalent default implementations.

The following table shows endpoint URLs for AM when configured as an OAuth 2.0 provider. For details, see the OAuth 2.0 Guide. The default endpoints are for Facebook as the OAuth 2.0 provider.

In addition to the endpoint URLs you can set other fields, like scope and attribute mapping, depending on the provider you use:

Endpoint URLs

AM Field	Details
Authorization Endpoint URL	`/oauth2/authorize` under the deployment URL.[a] Example: `https://openam.example.com:8443/openam/oauth2/realms/root/authorize`.
Access Token Endpoint URL	`/oauth2/access_token` under the deployment URL.[a] Example: `https://openam.example.com:8443/openam/oauth2/realms/root/access_token`.
User Profile Service URL	`/oauth2/tokeninfo` under the deployment URL. Example: `https://openam.example.com:8443/openam/oauth2/realms/root/tokeninfo`.

[a] This AM endpoint can take additional parameters. In particular, you must specify the realm if the AM OAuth 2.0 provider is configured for a subrealm rather than the top-level realm.

When making a REST API call, specify the realm in the path component of the endpoint. You must specify the entire hierarchy of the realm, starting at the top-level realm. Prefix each realm in the hierarchy with the `realms/` keyword. For example `/realms/root/realms/`*`customers`*`/realms/`*`europe`*.

For example, if the OAuth 2.0 provider is configured for the subrealm `customers` within the top-level realm, then the authentication endpoint URL is as follows: `https://openam.example.com:8443/openam/oauth2/realms/root/realms/customers/authorize`

The `/oauth2/authorize` endpoint can also take `module` and `service` parameters. Use either as described in " Authenticating From a Browser", where `module` specifies the authentication module instance to use or `service` specifies the authentication chain to use when authenticating the resource owner.

OAuth 2.0 Mix-Up Mitigation

AM has added a new property to the OAuth 2.0 authentication module, `openam-auth-oauth-mix-up-mitigation-enabled`. This OAuth 2.0 Mix-Up Mitigation property controls whether the OAuth 2.0 authentication module carries out additional verification steps when it receives the authorization code from the authorization server. This setting should be only enabled when the authorization server also supports OAuth 2.0 Mix-Up Mitigation.

OAuth 2.0 Mix-Up Mitigation Enabled

Specifies that the client must compare the issuer identifier of the authorization server upon registration with the issuer value returned in the `iss` response parameter. If they do not match, the client must abort the authorization process. The client must also confirm that the authorization server's response is intended for the client by comparing the client's client identifier to the value of the `client_id` response parameter.

For more information, see section 4 of OAuth 2.0 Mix-Up Mitigration Draft.

> **Note**
>
> At the time of this release, Facebook, Google, and Microsoft identity providers do not support this draft.

amster attribute: `mixUpMitigation`

ssoadm attribute: `openam-auth-oauth-mix-up-mitigation-enabled`

On the AM console, the field Token Issuer must be provided when the OAuth 2.0 Mix-Up Mitigation feature is enabled. The authorization code response will contain an issuer value (`iss`) that will be validated by the client. When the module is an OAuth2-only module (that is, OIDC is not used), the issuer value needs to be explicitly set in the Token Issuer field, so that the validation can succeed.

> **Note**
>
> Consult with the authorization server's documentation on what value it uses for the issuer field.

MSISDN Authentication Module Properties

amster service name: `MsisdnModule`

ssoadm service name: `sunAMAuthMSISDNService`

Trusted Gateway IP Address

Specifies a list of IP addresses of trusted clients that can access MSISDN modules. Either restrict the clients allowed to access the MSISDN module by adding each IPv4 or IPv6 address here, or leave the list empty to allow all clients to access the module. If you specify the value `none`, no clients are allowed access.

amster attribute: `trustedGatewayIPAddresses`

ssoadm attribute: `sunAMAuthMSISDNTrustedGatewayList`

MSISDN Number Search Parameter Name

Specifies a list of parameter names that identify which parameters to search in the request header or cookie header for the MSISDN number. For example, if you define x-Cookie-Param, AM_NUMBER, and COOKIE-ID, the MSISDN authentication service checks those parameters for the MSISDN number.

amster attribute: `msisdnParameterNames`

ssoadm attribute: `sunAMAuthMSISDNParameterNameList`

LDAP Server and Port

Specifies the LDAP server FQDN and its port in the format `ldap_server:port`. AM servers can be paired with LDAP servers and ports by adding entries with the format `AM_server|ldap_server:port`, for example, `openam.example.com|ldap1.example.com:649`.

To use SSL or TLS for security, enable the SSL/TLS Access to LDAP property. Make sure that AM can trust the servers' certificates when using this option.

amster attribute: `ldapProviderUrl`

ssoadm attribute: `sunAMAuthMSISDNLdapProviderUrl`

LDAP Start Search DN

Specifies the DN of the entry where the search for the user's MSISDN number should start. AM servers can be paired with search base DNs by adding entries with the format `AM_server|base_dn`. For example, `openam.example.com|dc=openam,dc=forgerock,dc=com` .

amster attribute: `baseSearchDN`

ssoadm attribute: `sunAMAuthMSISDNBaseDn`

Attribute To Use To Search LDAP

Specifies the name of the attribute in the user's profile that contains the MSISDN number to search for the user. The default is `sunIdentityMSISDNNumber`.

amster attribute: `userProfileMsisdnAttribute`

ssoadm attribute: `sunAMAuthMSISDNUserSearchAttribute`

LDAP Server Authentication User, LDAP Server Authentication Password

Specifies the bind DN and password to authenticate to the directory server. The default is `cn=Directory Manager`.

ssoadm attribute: `sunAMAuthMSISDNPrincipalUser` and `sunAMAuthMSISDNPrincipalPasswd`.

SSL/TLS for LDAP Access

When enabled, AM uses LDAPS or StartTLS to connect to the directory server. If you choose to enable SSL or TLS, then make sure that AM can trust the servers' certificates.

amster attribute: `ldapSslEnabled`

ssoadm attribute: `sunAMAuthMSISDNUseSsl`

MSISDN Header Search Attribute

Specifies which elements are searched for the MSISDN number.The possible values are:

`searchCookie`

To search the cookie.

`searchRequest`

To search the request header.

`searchParam`

To search the request parameters.

amster attribute: `msisdnRequestSearchLocations`

ssoadm attribute: `sunAMAuthMSISDNHeaderSearch`

LDAP Attribute Used to Retrieve User Profile

Specify the LDAP attribute that is used during a search to return the user profile for MSISDN authentication service. The default is `uid`.

amster attribute: `msisdnUserNamingAttribute`

ssoadm attribute: `sunAMAuthMSISDNUserNamingAttribute`

Return User DN to DataStore

When enabled, this option allows the authentication module to return the DN instead of the User ID. AM thus does not need to perform an additional search with the user ID to find the user's entry.

Enable this option only when the AM directory is the same as the directory configured for MSISDN searches.

amster attribute: `returnUserDN`

ssoadm attribute: `sunAMAuthMSISDNReturnUserDN`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

amster attribute: `authenticationLevel`

ssoadm attribute: `sunAMAuthMSISDNAuthLevel`

OATH Authentication Module Properties

amster service name: `OathModule`

ssoadm service name: `iPlanetAMAuthOATHService`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

amster attribute: `authenticationLevel`

ssoadm attribute: `iplanet-am-auth-oath-auth-level`

One Time Password Length

Sets the length of the OTP to six digits or longer. The default value is six.

amster attribute: `passwordLength`

ssoadm attribute: `iplanet-am-auth-oath-password-length`

Minimum Secret Key Length

The minimum number of hexadecimal characters allowed for the secret key.

amster attribute: `minimumSecretKeyLength`

ssoadm attribute: `iplanet-am-auth-oath-min-secret-key-length`

Secret Key Attribute Name

The name of the attribute where the key will be stored in the user profile.

amster attribute: `secretKeyAttribute`

ssoadm attribute: `iplanet-am-auth-oath-secret-key-attribute`

OATH Algorithm to Use

Select whether to use HOTP or TOTP. You can create an authentication chain to allow for a greater variety of devices. The default value is HOTP.

amster attribute: `oathAlgorithm`

ssoadm attribute: `iplanet-am-auth-oath-algorithm`

HOTP Window Size

The window that the OTP device and the server counter can be out of sync. For example, if the window size is 100 and the server's last successful login was at counter value 2, then the server will accept an OTP from device counter 3 to 102. The default value is 100.

amster attribute: `hotpWindowSize`

ssoadm attribute: `iplanet-am-auth-oath-hotp-window-size`

> **Note**
>
> For information on resetting the HOTP counter, see "Resetting Registered Devices by using REST".

Counter Attribute Name

The name of the HOTP attribute where the counter will be stored in the user profile.

amster attribute: `hotpCounterAttribute`

ssoadm attribute: `iplanet-am-auth-oath-hotp-counter-attribute`

Add Checksum Digit

Adds a checksum digit at the end of the HOTP password to verify the OTP was generated correctly. This is in addition to the actual password length. Set this only if your device supports it. The default value is No.

amster attribute: `addChecksum`

ssoadm attribute: `iplanet-am-auth-oath-add-checksum`

Truncation Offset

Advanced feature that is device-specific. Let this value default unless you know your device uses a truncation offset. The default value is -1.

amster attribute: `truncationOffset`

ssoadm attribute: `iplanet-am-auth-oath-truncation-offset`

TOTP Time Step Interval

The time interval for which an OTP is valid. For example, if the time step interval is 30 seconds, a new OTP will be generated every 30 seconds, and an OTP will be valid for 30 seconds. The default value is 30 seconds.

amster attribute: `timeStepSize`

ssoadm attribute: `iplanet-am-auth-oath-size-of-time-step`

One Time Password Max Retry

The number of times entry of the OTP may be attempted. Minimum is 1, maximum is 10.

Default: 3

amster attribute: `oathOtpMaxRetry`

ssoadm attribute: `forgerock-oath-max-retry`

TOTP Time Steps

The number of time step intervals that the system and the device can be off before password resynchronization is required. For example, if the number of TOTP time steps is 2 and the TOTP time step interval is 30 seconds, the server will allow an 89 second clock skew between the client and the server—two 30 second steps plus 29 seconds for the interval in which the OTP arrived. The default value is 2.

amster attribute: `stepsInWindow`

ssoadm attribute: `iplanet-am-auth-oath-steps-in-window`

Last Login Time Attribute

The name of the attribute where both HOTP and TOTP authentication will store information on when a person last logged in.

amster attribute: `lastLoginTimeAttribute`

ssoadm attribute: `iplanet-am-auth-oath-last-login-time-attribute-name`

The Shared Secret Provider Class

The class that processes the user profile attribute where the user's secret key is stored. The name of this attribute is specified in the Secret Key Attribute Name property.

Default: `org.forgerock.openam.authentication.modules.oath.plugins.DefaultSharedSecretProvider`

ssoadm attribute: `forgerock-oath-sharedsecret-implementation-class`

Clock Drift Attribute Name

The user profile attribute where the clock drift is stored. If this field is not specified, then AM does not check for clock drift.

ssoadm attribute: `forgerock-oath-observed-clock-drift-attribute-name`

Maximum Allowed Clock Drift

The maximum acceptable clock drift before authentication fails. If this value is exceeded, the user must register their device again.

The Maximum Allowed Clock Drift value should be greater than the TOTP Time Steps value.

ssoadm attribute: `forgerock-oath-maximum-clock-drift`

OpenID Connect id_token bearer Authentication Module Properties

The default settings are for Google's provider.

amster service name: `SocialAuthOpenIDModule`

ssoadm service name: `amAuthOpenIdConnect`

Account provider class

The account provider provides the means to search for and create OpenID Connect users given a set of attributes.

Default: `org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider`

amster attribute: `accountProviderClass`

ssoadm attribute: `openam-auth-openidconnect-account-provider-class`

OpenID Connect validation configuration type

In order to validate the ID token from the OpenID Connect provider, the module needs either a URL to get the public keys for the provider, or the symmetric key for an ID token signed with a HMAC-based algorithm.

By default, the configuration type is `.well-known/openid-configuration_url`. This means the module should retrieve the keys based on information in the OpenID Connect Provider Configuration Document.

You can instead configure the authentication module to validate the ID token signature with the client secret key you provide, or to validate the ID token with the keys retrieved from the URL to the OpenID Connect provider's JSON web key set.

> **Note**
>
> AM does not support non-string JWT header parameters, such as `jku` and `jwe`:
>
> - AM 6.5.2 and later parse the JWT but ignores the non-string header parameters.
> - AM versions earlier than 6.5.2 do not parse the JWT, and will return an HTTP 500 error message.
>
> Configure the public keys/certificates in AM instead of adding the headers, as explained in the relevant sections of the documentation.

`.well-known/openid-configuration_url` (Default)

Retrieve the provider keys based on the information provided in the OpenID Connect Provider Configuration Document.

Specify the URL to the document as the discovery URL.

`client_secret`

Use the client secret that you specify as the key to validate the ID token signature according to the HMAC, using the client secret to the decrypt the hash and then checking that the hash matches the hash of the ID token JWT.

`jwk_url`

Retrieve the provider's JSON web key set at the URL that you specify.

amster attribute: `cryptoContextType`

ssoadm attribute: `openam-auth-openidconnect-crypto-context-type`

OpenID Connect validation configuration value

Specifies the discovery URL, JWK or the client secret corresponding to the configuration type selected in the OpenID Connect validation configuration type property.

amster attribute: `cryptoContextValue`

ssoadm attribute: `openam-auth-openidconnect-crypto-context-value`

Name of header referencing the ID Token

Specifies the name of the HTTP request header to search for the ID token.

Default: `oidc_id_token`

amster attribute: `idTokenHeaderName`

ssoadm attribute: `openam-auth-openidconnect-header-name`

Name of OpenID Connect ID Token Issuer

Corresponds to the expected issue identifier value in the `iss` field of the ID token.

Default: `accounts.google.com`

amster attribute: `idTokenIssuer`

ssoadm attribute: `openam-auth-openidconnect-issuer-name`

Mapping of jwt attributes to local LDAP attributes

Maps OpenID Connect ID token claims to local user profile attributes, allowing the module to retrieve the user profile based on the ID token.

In OpenID Connect, an ID token is represented as a JSON Web Token (JWT). The ID Token section of the OpenID Connect Core 1.0 specification defines a number of claims included in the ID token for all flows. Additional claims depend on the scopes requested of the OpenID Connect provider.

For each item in the map, the key is the ID token field name and the value is the local user profile attribute name.

Default: `mail=email`, `uid=sub`

ssoadm attribute: `openam-auth-openidconnect-jwt-to-local-attribute-mappings`

Audience name

Specifies a case-sensitive audience name for this OpenID Connect authentication module. Used to check that the ID token received is intended for this module as an audience.

Default: `example`

amster attribute: `audienceName`

ssoadm attribute: `openam-auth-openidconnect-audience-name`

List of accepted authorized parties

Specifies a list of case-sensitive strings and/or URIs from which this authentication module accepts ID tokens. This list is checked against the authorized party claim of the ID token.

Default: `AuthorizedPartyExample` `http://www.example.com/authorized/party`

amster attribute: `acceptedAuthorizedParties`

ssoadm attribute: `openam-auth-openidconnect-accepted-authorized-parties`

Principal Mapper class

Specifies the class that implements the mapping of the OpenID Connect end user to an AM account. The default principal mapper uses the mapping of local attributes to ID token attributes to find a user profile.

Default: `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper`

amster attribute: `principalMapperClass`

ssoadm attribute: `openam-auth-openidconnect-principal-mapper-class`

Persistent Cookie Authentication Module Properties

amster service name: `PersistentCookieModule`

ssoadm service name: `iPlanetAMAuthPersistentCookieService`

Idle Timeout

Specifies the maximum idle time between requests in hours. If that time is exceeded, the cookie is no longer valid.

ssoadm attribute: `openam-auth-persistent-cookie-idle-time`

Max Life

Specifies the maximum life of the cookie in hours.

ssoadm attribute: `openam-auth-persistent-cookie-max-life`

Enforce Client IP

When enabled, enforces that the persistent cookie can only be used from the same client IP to which the cookie was issued.

ssoadm attribute: `openam-auth-persistent-cookie-enforce-ip`

Use secure cookie

When enabled, adds the "Secure" attribute to the persistent cookie.

ssoadm attribute: `openam-auth-persistent-cookie-secure-cookie`

Use HTTP only cookie

When enabled, adds the `HttpOnly` attribute to the persistent cookie.

ssoadm attribute: `openam-auth-persistent-cookie-http-only-cookie`

HMAC Signing Key

Specifies a key to use for HMAC signing of the persistent cookie. Values must be base64-encoded and at least 256 bits (32 bytes) long.

For example, to generate an HMAC signing key, run the following:

```
openssl rand -base64 32
```

or

```
cat /dev/urandom | tr -dc 'a-zA-Z0-9' | fold -w 32 | head -n 1|base64
```

Default: a random 256-bit secret key.

ssoadm attribute: `openam-auth-persistent-cookie-hmac-key`

RADIUS Authentication Module Properties

amster service name: `RadiusModule`

ssoadm service name: `iPlanetAMAuthRadiusService`

Primary Radius Servers, Secondary Radius Servers

Specify one or more primary and secondary RADIUS servers.

When configuring RADIUS servers, specify their IP address or FQDN. Configuring multiple servers allows you to map a RADIUS server to a specific AM instance in the form of *`AM_instance | RADIUS_server`*, where the AM instance is also specified by its IP address or FQDN.

Tip

Ensure each RADIUS server listens to the port specified in the Port Number field.

When authenticating users from a directory server that is remote to AM, set the primary values and, optionally, the secondary server values. Assuming a multi-data center environment, AM determines priority within the primary and secondary remote servers, respectively, as follows:

- Every RADIUS server that is mapped to the current AM instance has highest priority.
- Every RADIUS server that was not specifically mapped to a given AM instance has the next highest priority.

- RADIUS servers that are mapped to different AM instances have the lowest priority.

ssoadm attribute: `primary is iplanet-am-auth-radius-server1`; secondary is `iplanet-am-auth-radius-server2`

Shared Secret

Specify the shared secret for RADIUS authentication. The shared secret should be as secure as a well-chosen password.

amster attribute: `sharedSecret`

ssoadm attribute: `iplanet-am-auth-radius-secret`

Port Number

Specify the RADIUS server port.

Default is 1645.

amster attribute: `serverPortNumber`

ssoadm attribute: `iplanet-am-auth-radius-server-port`

Timeout

Specify how many seconds to wait for the RADIUS server to respond. The default value is 3 seconds.

amster attribute: `serverTimeout`

ssoadm attribute: `iplanet-am-auth-radius-timeout`

Health Check Interval

Used for failover. Specify how often AM performs a health check on a previously unavailable RADIUS server by sending an invalid authentication request.

Default: 5 minutes

amster attribute: `healthCheckInterval`

ssoadm attribute: `openam-auth-radius-healthcheck-interval`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

amster attribute: `authenticationLevel`

ssoadm attribute: `iplanet-am-auth-radius-auth-level`

SAE Authentication Module Properties

amster service name: `SaeModule`

ssoadm service name: `sunAMAuthSAEService`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

ssoadm service name: `sunAMAuthSAEAuthLevel`

SAML2 Authentication Module Properties

amster service name: `Saml2Module`

ssoadm service name: `iPlanetAMAuthSAML2Service`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

ssoadm attribute: `iplanet-am-auth-saml2-auth-level`

IdP Entity ID

Specifies the identity provider (IdP) for authentication requests to this module. Specify the name of a SAML v2.0 entity provider that is defined in the SAML2 authentication module's realm.

You can find configured entity providers in the AM console under Federation. The Realm column identifies the realm in which an entity provider has been configured.

amster attribute: `entityName`

ssoadm attribute: `forgerock-am-auth-saml2-entity-name`

SP MetaAlias

Specifies the local alias for the service provider (SP).

For service providers configured in the Top Level Realm, use the format /*SP Name*.

For service providers configured in subrealms, use the format /*Realm Name*/*SP Name*.

To find the local aliases for entity providers in the AM console, navigate to Realms > *Realm Name* > Applications > Federation > Entity Providers > *Entity Provider Name* > Services.

amster attribute: `metaAlias`

ssoadm attribute: `forgerock-am-auth-saml2-meta-alias`

Allow IdP to Create NameID

Specifies whether the IdP should create a new identifier for the authenticating user if none exists.

A value of `true` permits the IdP to create an identifier for the authenticating user if none exists. A value of `false` indicates a request to constrain the IdP from creating an identifier.

For detailed information, see the section on the `AllowCreate` property in SAML Version 2.0 Errata 05.

Default: `true`

amster attribute: `allowCreate`

ssoadm attribute: `forgerock-am-auth-saml2-allow-create`

Linking Authentication Chain

Specifies an authentication chain that is invoked when a user requires authentication to the SP.

Authentication to the SP is required when the authentication module running on the SP is unable to determine the user's identity based on the assertion received from the IdP. In this case, the linking authentication chain is invoked to allow the end user to link their remote and local accounts.

amster attribute: `loginChain`

ssoadm attribute: `forgerock-am-auth-saml2-login-chain`

Comparison Type

Specifies a comparison method to evaluate authentication context classes or statements. The value specified in this property overrides the value set in the SP configuration under Realms > *Realm Name* > Applications > Federation > Entity Providers > *Service Provider Name* > Assertion Content > Authentication Context > Comparison Type.

Valid comparison methods are `exact`, `minimum`, `maximum`, or `better`.

For more information about the comparison methods, see the section on the `<RequestedAuthnContext>` element in Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0.

Default: `exact`

amster attribute: `authComparison`

ssoadm attribute: `forgerock-am-auth-saml2-auth-comparison`

Authentication Context Class Reference

Specifies one or more URIs for authentication context classes to be included in the SAML request. Authentication context classes are unique identifiers for an authentication mechanism. The SAML v2.0 protocol supports a standard set of authentication context classes, defined in Authentication Context for the OASIS Security Assertion Markup Language (SAML) V2.0. In addition to the standard authentication context classes, you can specify customized authentication context classes.

Any authentication context class that you specify in this field must be supported for the service provider. To determine which authentication context classes are supported, locate the list of authentication context classes that are available to the SP under Realms > *Realm Name* > Applications > Federation > Entity Providers > *Service Provider Name* > Assertion Content > Authentication Context, and then review the values in the Supported column.

When specifying multiple authentication context classes, use the **|** character to separate the classes.

Example value: `urn:oasis:names:tc:SAML:2.0:ac:classes:Password|urn:oasis:names:tc:SAML:2.0:ac:classes:TimesyncToken`

amster attribute: `authnContextClassRef`

ssoadm attribute: `forgerock-am-auth-saml2-authn-context-class-ref`

Authentication Context Declaration Reference

Specifies one or more URIs that identify authentication context declarations.

This field is optional.

When specifying multiple URIs, use the **|** character to separate the URIs.

For more information, see the section on the `<RequestedAuthnContext>` element in Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0.

amster attribute: `authnContextDeclRef`

ssoadm attribute: `forgerock-am-auth-saml2-authn-context-decl-ref`

Request Binding

Specifies the format used to send the authentication request from the SP to the IdP.

Valid values are `HTTP-Redirect` and `HTTP-POST`.

Default: `HTTP-Redirect`

ssoadm attribute: `forgerock-am-auth-saml2-req-binding`. When using the **ssoadm** command, set this attribute's value to `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect` or `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST`.

Response Binding

Specifies the format used to send the response from the IdP to the SP.

A value of `HTTP-POST` indicates that the HTTP POST binding with a self-submitting form should be used in assertion processing. A value of `HTTP-Artifact` indicates that the HTTP Artifact binding should be used.

Default: `HTTP-Artifact`

ssoadm attribute: `forgerock-am-auth-saml2-binding`. When using the **ssoadm** command, set this attribute's value to `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact` or `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST`.

Force IdP Authentication

Specifies whether the IdP should force authentication or can reuse existing security contexts.

A value of `true` indicates that the IdP should force authentication. A value of `false` indicates that the IdP can reuse existing security contexts.

amster attribute: `forceAuthn`

ssoadm attribute: `forgerock-am-auth-saml2-force-authn`

Passive Authentication

Specifies whether the IdP should use passive authentication or not. Passive authentication requires the IdP to only use authentication methods that do not require user interaction. For example, authenticating using an X.509 certificate.

A value of `true` indicates that the IdP should authenticate passively. A value of `false` indicates that the IdP should not authenticate passively.

amster attribute: `isPassive`

ssoadm attribute: `forgerock-am-auth-saml2-is-passive`

NameID Format

Specifies a SAML name ID format to be requested in the SAML authentication request.

Default: `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent`

amster attribute: `nameIdFormat`

ssoadm attribute: `forgerock-am-auth-saml2-name-id-format`

Single Logout Enabled

Specifies whether AM should attempt to log out of the user's IdP session during session logout.

When enabling SAML v2.0 single logout, you must also configure the post-authentication processing class for the authentication chain containing the SAML2 authentication module to `org.forgerock.openam.authentication.modules.saml2.SAML2PostAuthenticationPlugin`.

For more information about configuring single logout when implementing SAML v2.0 federation using the SAML2 authentication module, see "Configuring Single Logout in an Integrated Mode Implementation" in the *SAML v2.0 Guide*.

Default: `false`

amster attribute: `sloEnabled`

ssoadm attribute: `forgerock-am-auth-saml2-slo-enabled`

Single Logout URL

Specifies the URL to which the user is forwarded after successful IdP logout. Configure this property only if you have enabled SAML v2.0 single logout by selecting the Single Logout Enabled check box.

amster attribute: `sloRelay`

ssoadm attribute: `forgerock-am-auth-saml2-slo-relay`

Scripted Authentication Module Properties

amster service name: `scripted`

ssoadm service name: `iPlanetAMAuthScriptedService`

Use the following settings at the realm level when configuring an individual scripted authentication module, in the AM console under Realms > *Realm Name* > Authentication > Modules.

Client-Side Script Enabled

When enabled, the module includes the specified client-side script in the login page to be executed on the user-agent prior to the server-side script.

amster attribute: `clientScriptEnabled`

ssoadm attribute: `iplanet-am-auth-scripted-client-script-enabled`

Client-Side Script

Specifies the ID of the script to include in the login page. This script is run on the user-agent prior to the server-side script. This script must be written in a language the user-agent can interpret, such as JavaScript, even if the server-side script is written in Groovy.

To create, view, or modify the content of the scripts, navigate to Realms > *Realm Name* > Scripts.

amster attribute: `clientScript`

ssoadm attribute: `iplanet-am-auth-scripted-client-script`

Server Side Script

Specifies the ID of the script to run in AM after the client-side script has completed.

To create, view, or modify the content of the scripts, navigate to Realms > *Realm Name* > Scripts.

amster attribute: `serverScript`

ssoadm attribute: `iplanet-am-auth-scripted-server-script`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the scripted authentication module.

The value can range from 0 to any positive integer.

amster attribute: `authenticationLevel`

ssoadm attribute: `iplanet-am-auth-scripted-auth-level`

In the AM console, navigate to Configure > Global Services > Scripting > Secondary Configurations > *Server-Side Script Type*,> Secondary Configurations > EngineConfiguration.

On the EngineConfiguration page, configure the following settings for the scripting engine of the selected type:

Server-side Script Timeout

Specifies the maximum execution time any individual script should take on the server (in seconds). AM terminates scripts which take longer to run than this value.

ssoadm attribute: `serverTimeout`

Core thread pool size

Specifies the initial number of threads in the thread pool from which scripts operate. AM will ensure the pool contains at least this many threads.

ssoadm attribute: `coreThreads`

Maximum thread pool size

Specifies the maximum number of threads in the thread pool from which scripts operate. If no free thread is available in the pool, AM creates new threads in the pool for script execution up to the configured maximum. It is recommended to set the maximum number of threads to 300.

ssoadm attribute: `maxThreads`

Thread pool queue size

Specifies the number of threads to use for buffering script execution requests when the maximum thread pool size is reached.

For short, CPU-bound scripts, consider a small pool size and larger queue length. For I/O-bound scripts, for example, REST calls, consider a larger maximum pool size and a smaller queue.

Not hot-swappable: restart server for changes to take effect.

ssoadm attribute: `queueSize`

Thread idle timeout (seconds)

Specifies the length of time (in seconds) for a thread to be idle before AM terminates created threads. If the current pool size contains the number of threads set in `Core thread pool size`, then idle threads will not be terminated, maintaining the initial pool size.

ssoadm attribute: `idleTimeout`

Java class whitelist

Specifies the list of class name patterns allowed to be invoked by the script. Every class accessed by the script must match at least one of these patterns.

You can specify the class name as-is or use a regular expression.

ssoadm attribute: `whiteList`

Java class blacklist

Specifies the list of class name patterns that are NOT allowed to be invoked by the script. The blacklist is applied AFTER the whitelist to exclude those classes. Access to a class specified in both the whitelist and the blacklist will be denied.

You can specify the class name to exclude as-is or use a regular expression.

ssoadm attribute: `blackList`

Use system SecurityManager

When enabled, AM makes a call to the `System.getSecurityManager().checkPackageAccess(...)` method for each class that is accessed. The method throws `SecurityException` if the calling thread is not allowed to access the package.

> **Note**
>
> This feature only takes effect if the security manager is enabled for the JVM.

ssoadm attribute: `useSecurityManager`

SecurID Authentication Module Properties

> **Important**
>
> To use the SecurID authentication module, you must first build an AM `.war` file that includes the supporting library. For more information, see "Enabling RSA SecurID Support" in the *Installation Guide*.

amster service name: `securid`

ssoadm service name: `iPlanetAMAuthSecurIDService`

ACE/Server Configuration Path

Specify the directory where the SecurID ACE/Server `sdconf.rec` file is located, which by default is expected under the AM configuration directory, such as `$HOME/openam/openam/auth/ace/data`. The directory must exist before AM can use SecurID authentication.

amster attribute: `serverConfigPath`

ssoadm attribute: `iplanet-am-auth-securid-server-config-path`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

amster attribute: `authenticationLevel`

ssoadm attribute: `iplanet-am-auth-securid-auth-level`

Social Authentication Module Properties - Instagram

amster service name: `SocialAuthInstagramModule`

ssoadm service name: `iPlanetAMAuthSocialAuthInstagramService`

Core

The following properties are available under the Core tab:

Authentication Level

Specifies the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

Default: `0`

amster data attribute: `authenticationLevel`

Social Provider

Specifies the name of the social provider for which this module is being set up.

Default: `Instagram`

amster data attribute: `provider`

Client Id

Specifies the `client_id` parameter as described in section 2.2 of The OAuth 2.0 Authorization Framework (RFC 6749).

> **Tip**
>
> To register an application with Instagram and obtain an OAuth 2.0 `client_id` and `client_secret`, visit https://www.instagram.com/developer/.

amster attribute: `clientId`

Client Secret

Specifies the `client_secret` parameter as described in section 2.3 of The OAuth 2.0 Authorization Framework (RFC 6749).

amster attribute: `clientSecret`

Authentication Endpoint URL

Specifies the URL to the social provider's endpoint handling authentication as described in section 3.1 of The OAuth 2.0 Authorization Framework (RFC 6749).

Default: `https://api.instagram.com/oauth/authorize`

amster attribute: `authorizeEndpoint`

Access Token Endpoint URL

Specifies the URL to the endpoint handling access tokens as described in section 3.2 of The OAuth 2.0 Authorization Framework (RFC 6749).

Default: `https://api.instagram.com/oauth/access_token`

amster attribute: `tokenEndpoint`

User Profile Service URL

Specifies the user profile URL that returns profile information in JSON format.

Default:`https://api.instagram.com/v1/users/self`

amster attribute: `userInfoEndpoint`

Scope

Specifies a list of user profile attributes that the client application requires, according to *The OAuth 2.0 Authorization Framework (RFC 6749)*. The list depends on the permissions that the resource owner, such as the end user, grants to the client application.

Default: `basic`

amster attribute: `scope`

Scope Delimiter

Specifies the delimiter used to separate scope values.

Some authorization servers use non-standard separators for scopes. Facebook, for example, uses commas.

Default: space character

amster attribute: `scopeDelimiter`

Subject Property

Specifies the attribute the social provider uses to identify a user.

Default: `id`

amster attribute: `subjectProperty`

Use Basic Auth

Specifies that the client uses HTTP Basic authentication when authenticating to the social provider.

Valid values are:

- `true`
- `false`

Default: `false`

amster attribute: `usesBasicAuth`

Proxy URL

Specifies the URL to the `/oauth2c/OAuthProxy.jsp` file, which provides AM with GET to POST proxying capabilities. Change this URL only if an external server performs the GET to POST proxying.

Default: `@SERVER_PROTO@://@SERVER_HOST@:@SERVER_PORT@/@SERVER_URI@/oauth2c/OAuthProxy.jsp`

Example: `https://openam.example.com:8443/openam/oauth2c/OAuthProxy.jsp`

amster attribute: `ssoProxyUrl`

OAuth 2.0 Provider Logout Service

Specifies the URL of the social provider's logout service.

To enable logout of the social authentication provider when logging out of AM, you must add `org.forgerock.openam.authentication.modules.oauth2.OAuth2PostAuthnPlugin` to the Authentication Post Processing Classes property. To add the class, navigate to Authentication > Settings > Post Authentication Processing.

Default: `https://instagram.com/accounts/logout`

amster attribute: `logoutServiceUrl`

Logout Options

Specifies the social provider logout actions to take when logging out of AM.

Valid options are:

`prompt`

Asks the user whether or not to log out from the social provider.

`logout`

Logs the user out of the social provider without prompting.

`donotlogout`

Keeps the user logged in to the social provider. There is no prompt to the user.

Default: `prompt`

amster attribute: `logoutBehaviour`

Account Provisioning

The following properties are available under the Account Provisioning tab:

Use IDM as Registration Service

Whether to use IDM as an external registration service to complete registration for new users. You must configure and enable the IDM Provisioning service to use this option. See "IDM Provisioning".

AM passes IDM these parameters:

- `clientToken`: Signed, encrypted JWT of the OAuth 2.0 authentication state.
- `returnParams`: Encoded URL parameters, required to be returned to AM to resume authentication after registration in IDM is complete.

Default: `False`

amster attribute: `enableRegistrationService`

Create account if it does not exist

When enabled, AM creates an account for the user if the user profile does not exist.

When disabled, a user without a profile may still log into AM if the Ignore Profile attribute is set in the authentication service of the realm, or if the account is mapped to an anonymous account.

Valid values are:

- `true`
- `false`

Default: `true`

amster attribute: `createAccount`

Account Provider

Specifies the name of the class that implements the account provider.

Default: `org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider`

amster attribute: `accountProviderClass`

Account Mapper

Specifies the name of the class that implements the attribute mapping for the account search.

> **Tip**
>
> You can provide string constructor parameters by appending pipe-separated (|) values.

Default: `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper|*|instagram-`

amster attribute: `accountMapperClass`

Account Mapper Configuration

Specifies the attribute configuration used to map the account of the user authenticated in the social provider to the local data store in AM. Valid values take the form *`provider-attr=local-attr`*.

> **Tip**
>
> When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings, by using dot notation.
>
> For example, given a JSON payload of:
>
> ```
> {
>   "sub" : "12345",
>   "name" : {
>     "first_name" : "Demo",
>     "last_name" : "User"
>   }
> }
> ```
>
> You can create a mapper such as:
>
> `name.first_name=cn`

Default: `id=uid`

amster attribute: `accountMapperConfiguration`

Attribute Mapper

Specifies the list of fully qualified class names for implementations that map attributes from the social provider to AM profile attributes.

You can provide a custom attribute mapper. A custom attribute mapper must implement the `org.forgerock.openam.authentication.modules.common.mapping.AttributeMapper` interface.

Provided implementations are:

- `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper`
- `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper` - can only be used when using the `openid` scope

Tip

You can provide string constructor parameters by appending pipe-separated (|) values.

For example, the `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper` class can take two constructor parameters: a comma-separated list of attributes, and a prefix to apply to their values. Specify these as follows:

`org.forgerock.openam.authentication.modules.oidc.JsonAttributeMapper|uid|instagram-`

Default: `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper|uid|instagram-`

amster attribute: `attributeMappingClasses`

Attribute Mapper Configuration

Specifies a map of social provider user account attributes to local user profile attributes with values in the form *`provider-attr=local-attr`*.

Tip

When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings, by using dot notation.

For example, given a JSON payload of:

```
{
  "sub" : "12345",
  "name" : {
    "first_name" : "Demo",
    "last_name" : "User"
  }
}
```

You can create a mapper such as:

`name.first_name=cn`

Default:

```
id=uid
full_name=sn
username=cn
username=givenName
```

amster attribute: `attributeMapperConfiguration`

Map to anonymous user

When enabled, maps the social provider authenticated user to a specified anonymous user. If the Create account if it does not exist property is enabled, AM creates an account for the authenticated user instead of mapping the account to an anonymous user.

Valid values are:

- `true`
- `false`

Default: `false`

amster attribute: `mapToAnonymousUser`

Anonymous User

Specifies an anonymous user that exists in the current realm. The Map to anonymous user property maps authorized users without a profile to this anonyomus user, if enabled.

Default: `anonymous`

amster attribute: `anonymousUserName`

Save attributes in the session

When enabled, saves the values of attributes specified in the Attribute Mapper Configuration property in the AM session.

Valid values are:

- `true`
- `false`

Default: `true`

amster attribute: `saveAttributesInSession`

Social Authentication Module Properties - OAuth 2.0

amster service name: `SocialAuthOAuth2Module`

ssoadm service name: `iPlanetAMAuthSocialAuthOAuth2Service`

Core

The following properties are available under the Core tab:

Authentication Level

Specifies the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

Default: `0`

amster data attribute: `authenticationLevel`

Social Provider

Specifies the name of the social provider for which this module is being set up.

Example: `Google`

amster data attribute: `provider`

Client Id

Specifies the `client_id` parameter as described in section 2.2 of The OAuth 2.0 Authorization Framework (RFC 6749).

amster attribute: `clientId`

Client Secret

Specifies the `client_secret` parameter as described in section 2.3 of The OAuth 2.0 Authorization Framework (RFC 6749).

amster attribute: `clientSecret`

Authentication Endpoint URL

Specifies the URL to the social provider's endpoint handling authentication as described in section 3.1 of The OAuth 2.0 Authorization Framework (RFC 6749).

Example: `https://accounts.google.com/o/oauth2/v2/auth`

amster attribute: `authorizeEndpoint`

Access Token Endpoint URL

Specifies the URL to the endpoint handling access tokens as described in section 3.2 of The OAuth 2.0 Authorization Framework (RFC 6749).

Example: `https://www.googleapis.com/oauth2/v4/token`

amster attribute: `tokenEndpoint`

User Profile Service URL

Specifies the user profile URL that returns profile information in JSON format.

Exaple: `https://www.googleapis.com/oauth2/v3/userinfo`

amster attribute: `userInfoEndpoint`

Scope

Specifies a list of user profile attributes that the client application requires, according to *The OAuth 2.0 Authorization Framework (RFC 6749)*. The list depends on the permissions that the resource owner, such as the end user, grants to the client application.

amster attribute: `scope`

Scope Delimiter

Specifies the delimiter used to separate scope values.

Some authorization servers use non-standard separators for scopes. Facebook, for example, uses commas.

amster attribute: `scopeDelimiter`

Subject Property

Specifies the attribute the social provider uses to identify a user.

Example: `sub`

amster attribute: `subjectProperty`

Use Basic Auth

Specifies that the client uses HTTP Basic authentication when authenticating to the social provider.

Valid values are:

- `true`
- `false`

Default: `true`

amster attribute: `usesBasicAuth`

Proxy URL

Specifies the URL to the `/oauth2c/OAuthProxy.jsp` file, which provides AM with GET to POST proxying capabilities. Change this URL only if an external server performs the GET to POST proxying.

Default: `@SERVER_PROTO@://@SERVER_HOST@:@SERVER_PORT@/@SERVER_URI@/oauth2c/OAuthProxy.jsp`

Example: `https://openam.example.com:8443/openam/oauth2c/OAuthProxy.jsp`

amster attribute: `ssoProxyUrl`

OAuth 2.0 Provider Logout Service

Specifies the URL of the social provider's logout service.

To enable logout of the social authentication provider when logging out of AM, you must add `org.forgerock.openam.authentication.modules.oauth2.OAuth2PostAuthnPlugin` to the Authentication Post Processing Classes property. To add the class, navigate to Authentication > Settings > Post Authentication Processing.

amster attribute: `logoutServiceUrl`

Logout Options

Specifies the social provider logout actions to take when logging out of AM.

Valid options are:

`prompt`

Asks the user whether or not to log out from the social provider.

`logout`

Logs the user out of the social provider without prompting.

`donotlogout`

Keeps the user logged in to the social provider. There is no prompt to the user.

Default: `prompt`

amster attribute: `logoutBehaviour`

Token Issuer

Corresponds to the expected issue identifier value in the `iss` field of the ID token.

Example: `https://accounts.google.com`

amster attribute: `issuerName`

OAuth 2.0 Mix-Up Mitigation Enabled

Controls whether the OAuth 2.0 authentication module carries out additional verification steps when it receives the authorization code from the authorization server.

Specifies that the client must compare the issuer identifier of the authorization server upon registration with the issuer value returned in the `iss` response parameter. If they do not match, the client must abort the authorization process. The client must also confirm that the authorization server's response is intended for the client by comparing the client's client identifier to the value of the `client_id` response parameter.

The Token Issuer property must be entered when the OAuth 2.0 Mix-Up Mitigation feature is enabled, so that the validation can succeed. The authorization code response will contain an issuer value (`iss`) that will be validated by the client.

> **Note**
>
> Consult with the authorization server's documentation on what value it uses for the issuer field.

For more information, see section 4 of OAuth 2.0 Mix-Up Mitigration Draft.

amster attribute: `mixUpMitigation`

Account Provisioning

The following properties are available under the Account Provisioning tab:

Use IDM as Registration Service

Whether to use IDM as an external registration service to complete registration for new users. You must configure and enable the IDM Provisioning service to use this option. See "IDM Provisioning".

AM passes IDM these parameters:

- `clientToken`: Signed, encrypted JWT of the OAuth 2.0 authentication state.
- `returnParams`: Encoded URL parameters, required to be returned to AM to resume authentication after registration in IDM is complete.

Default: `False`

amster attribute: `enableRegistrationService`

Create account if it does not exist

When enabled, AM creates an account for the user if the user profile does not exist. If the Prompt for password setting and activation code attribute is enabled, AM prompts the user for a password and activation code before creating the account.

> **Important**
>
> When configured to create new accounts, the SMTP settings must also be valid. As part of account creation, the authentication module sends the resource owner an email with an account activation code. To send the mail, AM uses the SMTP settings you provide in the module configuration.

When disabled, a user without a profile may still log into AM if the Ignore Profile attribute is set in the authentication service of the realm, or if the account is mapped to an anonymous account.

Valid values are:

- `true`
- `false`

Default: `true`

amster attribute: `createAccount`

Account Provider

Specifies the name of the class that implements the account provider.

Default: `org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider`

amster attribute: `accountProviderClass`

Account Mapper

Specifies the name of the class that implements the attribute mapping for the account search.

Example: `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper|*|google-`

amster attribute: `accountMapperClass`

Account Mapper Configuration

Specifies the attribute configuration used to map the account of the user authenticated in the social provider to the local data store in AM. Valid values take the form *`provider-attr=local-attr`*.

> **Tip**
>
> When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings, by using dot notation.

For example, given a JSON payload of:

```
{
  "sub" : "12345",
  "name" : {
    "first_name" : "Demo",
    "last_name" : "User"
  }
}
```

You can create a mapper such as:

`name.first_name=cn`

amster attribute: `accountMapperConfiguration`

Attribute Mapper

Specifies the list of fully qualified class names for implementations that map attributes from the social provider to AM profile attributes.

You can provide a custom attribute mapper. A custom attribute mapper must implement the `org.forgerock.openam.authentication.modules.common.mapping.AttributeMapper` interface.

Provided implementations are:

- `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper`
- `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper` - can only be used when using the `openid` scope

Tip

You can provide string constructor parameters by appending pipe-separated (|) values.

For example, the `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper` class can take two constructor parameters: a comma-separated list of attributes, and a prefix to apply to their values. Specify these as follows:

`org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper|*|google-`

amster attribute: `attributeMappingClasses`

Attribute Mapper Configuration

Specifies a map of social provider user account attributes to local user profile attributes with values in the form *`provider-attr=local-attr`*.

Tip

When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings, by using dot notation.

For example, given a JSON payload of:

```
{
  "sub" : "12345",
  "name" : {
    "first_name" : "Demo",
    "last_name" : "User"
  }
}
```

You can create a mapper such as:

`name.first_name=cn`

amster attribute: `attributeMapperConfiguration`

Prompt for password setting and activation code

When enabled, the user must set a password before AM creates an account dynamically. An activation code is also sent to the user's email address. Both the password and the code are required before the account is created.

Valid values are:

- `true`
- `false`

Default: `false`

amster attribute: `promptPasswordFlag`

Map to anonymous user

When enabled, maps the social provider authenticated user to a specified anonymous user. If the Create account if it does not exist property is enabled, AM creates an account for the authenticated user instead of mapping the account to an anonymous user.

Valid values are:

- `true`
- `false`

Default: `false`

amster attribute: `mapToAnonymousUser`

Anonymous User

Specifies an anonymous user that exists in the current realm. The Map to anonymous user property maps authorized users without a profile to this anonyomus user, if enabled.

Default: `anonymous`

amster attribute: `anonymousUserName`

Save attributes in the session

When enabled, saves the values of attributes specified in the Attribute Mapper Configuration property in the AM session.

Valid values are:

- `true`
- `false`

Default: `true`

amster attribute: `saveAttributesInSession`

Email

The following properties are available under the Email tab:

Email attribute in the Response

Specifies the attribute identifying the authenticated user's email address in the response from the profile service in the social provider. This setting is used to send an email message with an activation code for accounts created dynamically.

amster attribute: `emailAttribute`

Mail Server Gateway implementation class

Specifies the class used by the module to send email. A custom subclass of `org.forgerock.openam.authentication.modules.oauth2.EmailGateway` class can be provided.

Default: `org.forgerock.openam.authentication.modules.oauth2.DefaultEmailGatewayImpl`

amster attribute: `emailGateway`

SMTP host

Specifies the host name of the mail server.

Default: `localhost`

amster attribute: `smtpHost`

SMTP port

Specifies the SMTP port number for the mail server.

Default: `25`

amster attribute: `smtpPort`

SMTP User Name

Specifies the username AM uses to authenticate to the mail server.

amster attribute: `smtpUsername`

SMTP User Password

Specifies the password AM uses to authenticate to the mail server.

amster attribute: `smtpPassword`

SMTP SSL Enabled

When enabled, connects to the mail server over SSL. AM must be able to trust the SMTP server certificate.

Valid values are:

- `true`
- `false`

Default: `false`

amster attribute: `smtpSslEnabled`

SMTP From address

Specifies the address of the email sender, such as `no-reply@example.com`.

amster attribute: `smtpFromAddress`

Social Authentication Module Properties - OpenID Connect 1.0

The example settings are for Google.

amster service name: `SocialAuthOpenIDModule`

ssoadm service name: `iPlanetAMAuthSocialAuthOpenIDService`

Core

The following properties are available under the Core tab:

Social Provider

Specifies the name of the social provider for which this module is being set up.

Example: `Google`

amster data attribute: `provider`

Client Id

Specifies the `client_id` parameter as described in section 2.2 of The OAuth 2.0 Authorization Framework (RFC 6749).

amster attribute: `clientId`

Client Secret

Specifies the `client_secret` parameter as described in section 2.3 of The OAuth 2.0 Authorization Framework (RFC 6749).

amster attribute: `clientSecret`

Authentication Level

Specifies the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

Default: `0`

amster data attribute: `authenticationLevel`

Authentication Endpoint URL

Specifies the URL to the social provider's endpoint handling authentication as described in section 3.1 of The OAuth 2.0 Authorization Framework (RFC 6749).

Example: `https://accounts.google.com/o/oauth2/v2/auth`

amster attribute: `authorizeEndpoint`

Access Token Endpoint URL

Specifies the URL to the endpoint handling access tokens as described in section 3.2 of The OAuth 2.0 Authorization Framework (RFC 6749).

Example: `https://www.googleapis.com/oauth2/v4/token`

amster attribute: `tokenEndpoint`

User Profile Service URL

Specifies the user profile URL that returns profile information in JSON format.

Exaple: `https://www.googleapis.com/oauth2/v3/userinfo`

amster attribute: `userInfoEndpoint`

Scope

Specifies a list of user profile attributes that the client application requires, according to *The OAuth 2.0 Authorization Framework (RFC 6749)*. The list depends on the permissions that the resource owner, such as the end user, grants to the client application.

Default: `openid`

amster attribute: `scope`

Scope Delimiter

Specifies the delimiter used to separate scope values.

Some authorization servers use non-standard separators for scopes. Facebook, for example, uses commas.

amster attribute: `scopeDelimiter`

Subject Property

Specifies the attribute the social provider uses to identify a user.

Example: `sub`

amster attribute: `subjectProperty`

Use Basic Auth

Specifies that the client uses HTTP Basic authentication when authenticating to the social provider.

Valid values are:

- `true`
- `false`

Default: `true`

amster attribute: `usesBasicAuth`

Proxy URL

Specifies the URL to the `/oauth2c/OAuthProxy.jsp` file, which provides AM with GET to POST proxying capabilities. Change this URL only if an external server performs the GET to POST proxying.

Default: `@SERVER_PROTO@://@SERVER_HOST@:@SERVER_PORT@/@SERVER_URI@/oauth2c/OAuthProxy.jsp`

Example: `https://openam.example.com:8443/openam/oauth2c/OAuthProxy.jsp`

amster attribute: `ssoProxyUrl`

OAuth 2.0 Provider Logout Service

Specifies the URL of the social provider's logout service.

To enable logout of the social authentication provider when logging out of AM, you must add `org.forgerock.openam.authentication.modules.oauth2.OAuth2PostAuthnPlugin` to the Authentication Post Processing Classes property. To add the class, navigate to Authentication > Settings > Post Authentication Processing.

amster attribute: `logoutServiceUrl`

Logout Options

Specifies the social provider logout actions to take when logging out of AM.

Valid options are:

`prompt`

Asks the user whether or not to log out from the social provider.

`logout`

Logs the user out of the social provider without prompting.

`donotlogout`

Keeps the user logged in to the social provider. There is no prompt to the user.

Default: `prompt`

amster attribute: `logoutBehaviour`

Token Issuer

Corresponds to the expected issue identifier value in the `iss` field of the ID token.

Example: `https://accounts.google.com`

amster attribute: `issuerName`

OAuth 2.0 Mix-Up Mitigation Enabled

Controls whether the OAuth 2.0 authentication module carries out additional verification steps when it receives the authorization code from the authorization server.

Specifies that the client must compare the issuer identifier of the authorization server upon registration with the issuer value returned in the `iss` response parameter. If they do not match, the client must abort the authorization process. The client must also confirm that the authorization server's response is intended for the client by comparing the client's client identifier to the value of the `client_id` response parameter.

The Token Issuer property must be entered when the OAuth 2.0 Mix-Up Mitigation feature is enabled, so that the validation can succeed. The authorization code response will contain an issuer value (`iss`) that will be validated by the client.

> **Note**
>
> Consult with the authorization server's documentation on what value it uses for the issuer field.

For more information, see section 4 of OAuth 2.0 Mix-Up Mitigration Draft.

amster attribute: `mixUpMitigation`

OpenID Connect

The following properties are available under the OpenID Connect tab:

OpenID Connect validation configuration type

In order to validate the ID token from the OpenID Connect provider, the module needs either a URL to get the public keys for the provider, or the symmetric key for an ID token signed with a HMAC-based algorithm.

By default, the configuration type is `.well-known/openid-configuration_url`. This means the module should retrieve the keys based on information in the OpenID Connect Provider Configuration Document.

You can instead configure the authentication module to validate the ID token signature with the client secret key you provide, or to validate the ID token with the keys retrieved from the URL to the OpenID Connect provider's JSON web key set.

`.well-known/openid-configuration_url` **(Default)**

Retrieve the provider keys based on the information provided in the OpenID Connect Provider Configuration Document.

Specify the URL to the document in the OpenID Connect validation configuration value property

`client_secret`

Use the client secret that you specify in the Client Secret property (not the OpenID Connect validation configuration value property, which is ignored) as the key to validate the ID token signature according to the HMAC, using the client secret to the decrypt the hash and then checking that the hash matches the hash of the ID token JWT.

`jwk_url`

Retrieve the provider's JSON web key set at the URL that you specify in the OpenID Connect validation configuration value property.

amster attribute: `cryptoContextType`

OpenID Connect validation configuration value

Specifies the full URL to the discovery or JWK location, corresponding to the configuration type selected in the OpenID Connect validation configuration type property.

Example: `https://accounts.google.com/.well-known/openid-configuration`

amster attribute: `cryptoContextValue`

Account Provisioning

The following properties are available under the Account Provisioning tab:

Use IDM as Registration Service

Whether to use IDM as an external registration service to complete registration for new users. You must configure and enable the IDM Provisioning service to use this option. See "IDM Provisioning".

AM passes IDM these parameters:

- `clientToken`: Signed, encrypted JWT of the OAuth 2.0 authentication state.
- `returnParams`: Encoded URL parameters, required to be returned to AM to resume authentication after registration in IDM is complete.

Default: `False`

amster attribute: `enableRegistrationService`

Create account if it does not exist

When enabled, AM creates an account for the user if the user profile does not exist. If the Prompt for password setting and activation code attribute is enabled, AM prompts the user for a password and activation code before creating the account.

> **Important**
>
> When configured to create new accounts, the SMTP settings must also be valid. As part of account creation, the authentication module sends the resource owner an email with an account activation code. To send the mail, AM uses the SMTP settings you provide in the module configuration.

When disabled, a user without a profile may still log into AM if the Ignore Profile attribute is set in the authentication service of the realm, or if the account is mapped to an anonymous account.

Valid values are:

- `true`
- `false`

Default: `true`

amster attribute: `createAccount`

Account Provider

Specifies the name of the class that implements the account provider.

Default: `org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider`

amster attribute: `accountProviderClass`

Account Mapper

Specifies the name of the class that implements the attribute mapping for the account search.

> **Tip**
>
> You can provide string constructor parameters by appending pipe-separated (|) values.

Example: `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper|*|google-`

amster attribute: `accountMapperClass`

Account Mapper Configuration

Specifies the attribute configuration used to map the account of the user authenticated in the social provider to the local data store in AM. Valid values take the form *provider-attr=local-attr*.

> **Tip**
>
> When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings, by using dot notation.
>
> For example, given a JSON payload of:
>
> ```
> {
>   "sub" : "12345",
>   "name" : {
>     "first_name" : "Demo",
>     "last_name" : "User"
>   }
> }
> ```
>
> You can create a mapper such as:
>
> `name.first_name=cn`

amster attribute: `accountMapperConfiguration`

Attribute Mapper

Specifies the list of fully qualified class names for implementations that map attributes from the social provider to AM profile attributes.

You can provide a custom attribute mapper. A custom attribute mapper must implement the `org.forgerock.openam.authentication.modules.common.mapping.AttributeMapper` interface.

Provided implementations are:

- `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper`
- `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper` - can only be used when using the `openid` scope

> **Tip**
>
> You can provide string constructor parameters by appending pipe-separated (|) values.
>
> For example, the `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper` class can take two constructor parameters: a comma-separated list of attributes, and a prefix to apply to their values. Specify these as follows:
>
> `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper|*|google-`

amster attribute: `attributeMappingClasses`

Attribute Mapper Configuration

Specifies a map of social provider user account attributes to local user profile attributes with values in the form *provider-attr=local-attr*.

> **Tip**
>
> When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings, by using dot notation.
>
> For example, given a JSON payload of:
>
> ```
> {
>   "sub" : "12345",
>   "name" : {
>     "first_name" : "Demo",
>     "last_name" : "User"
>   }
> }
> ```
>
> You can create a mapper such as:
>
> `name.first_name=cn`

amster attribute: `attributeMapperConfiguration`

Prompt for password setting and activation code

When enabled, the user must set a password before AM creates an account dynamically. An activation code is also sent to the user's email address. Both the password and the code are required before the account is created.

Valid values are:

- `true`
- `false`

Default: `false`

amster attribute: `promptPasswordFlag`

Map to anonymous user

When enabled, maps the social provider authenticated user to a specified anonymous user. If the Create account if it does not exist property is enabled, AM creates an account for the authenticated user instead of mapping the account to an anonymous user.

Valid values are:

- `true`

- `false`

Default: `false`

amster attribute: `mapToAnonymousUser`

Anonymous User

Specifies an anonymous user that exists in the current realm. The Map to anonymous user property maps authorized users without a profile to this anonyomus user, if enabled.

Default: `anonymous`

amster attribute: `anonymousUserName`

Save attributes in the session

When enabled, saves the values of attributes specified in the Attribute Mapper Configuration property in the AM session.

Valid values are:

- `true`
- `false`

Default: `true`

amster attribute: `saveAttributesInSession`

Email

The following properties are available under the Email tab:

Email attribute in the Response

Specifies the attribute identifying the authenticated user's email address in the response from the profile service in the social provider. This setting is used to send an email message with an activation code for accounts created dynamically.

amster attribute: `emailAttribute`

Mail Server Gateway implementation class

Specifies the class used by the module to send email. A custom subclass of `org.forgerock.openam.authentication.modules.oauth2.EmailGateway` class can be provided.

Default: `org.forgerock.openam.authentication.modules.oauth2.DefaultEmailGatewayImpl`

amster attribute: `emailGateway`

SMTP host

Specifies the host name of the mail server.

Default: `localhost`

amster attribute: `smtpHost`

SMTP port

Specifies the SMTP port number for the mail server.

Default: `25`

amster attribute: `smtpPort`

SMTP User Name

Specifies the username AM uses to authenticate to the mail server.

amster attribute: `smtpUsername`

SMTP User Password

Specifies the password AM uses to authenticate to the mail server.

amster attribute: `smtpPassword`

SMTP SSL Enabled

When enabled, connects to the mail server over SSL. AM must be able to trust the SMTP server certificate.

Valid values are:

- `true`
- `false`

Default: `false`

amster attribute: `smtpSslEnabled`

SMTP From address

Specifies the address of the email sender, such as `no-reply@example.com`.

amster attribute: `smtpFromAddress`

Social Authentication Module Properties - VKontakte

amster service name: `SocialAuthVKontakteModule`

ssoadm service name: `iPlanetAMAuthSocialAuthVKService`

Core

The following properties are available under the Core tab:

Social Provider

Specifies the name of the social provider for which this module is being set up.

Default: `VKontakte`

amster data attribute: `provider`

Client Id

Specifies the `client_id` parameter as described in section 2.2 of The OAuth 2.0 Authorization Framework (RFC 6749).

> **Tip**
>
> To register an application with VKontakte and obtain an OAuth 2.0 `client_id` and `client_secret`, visit https://vk.com/apps?act=manage.

amster attribute: `clientId`

Client Secret

Specifies the `client_secret` parameter as described in section 2.3 of The OAuth 2.0 Authorization Framework (RFC 6749).

amster attribute: `clientSecret`

Authentication Level

Specifies the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

Default: `0`

amster data attribute: `authenticationLevel`

Authentication Endpoint URL

Specifies the URL to the endpoint handling authentication as described in section 3.1 of The OAuth 2.0 Authorization Framework (RFC 6749).

Default: `https://oauth.vk.com/authorize`

amster attribute: `authorizeEndpoint`

Access Token Endpoint URL

Specifies the URL to the social provider's endpoint handling access tokens as described in section 3.2 of The OAuth 2.0 Authorization Framework (RFC 6749).

Default: `https://oauth.vk.com/access_token`

amster attribute: `tokenEndpoint`

User Profile Service URL

Specifies the user profile URL that returns profile information in JSON format.

Default:`https://api.vk.com/method/users.get`

amster attribute: `userInfoEndpoint`

Scope

Specifies a list of user profile attributes that the client application requires, according to *The OAuth 2.0 Authorization Framework (RFC 6749)*. The list depends on the permissions that the resource owner, such as the end user, grants to the client application.

amster attribute: `scope`

Proxy URL

Specifies the URL to the `/oauth2c/OAuthProxy.jsp` file, which provides AM with GET to POST proxying capabilities. Change this URL only if an external server performs the GET to POST proxying.

Default: `@SERVER_PROTO@://@SERVER_HOST@:@SERVER_PORT@/@SERVER_URI@/oauth2c/OAuthProxy.jsp`

Example: `https://openam.example.com:8443/openam/oauth2c/OAuthProxy.jsp`

amster attribute: `ssoProxyUrl`

Subject Property

Specifies the attribute the social provider uses to identify a user.

Default: `id`

amster attribute: `subjectProperty`

Account Provisioning

The following properties are available under the Account Provisioning tab:

Account Provider

Specifies the name of the class that implements the account provider.

Default: `org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider`

amster attribute: `accountProviderClass`

Use IDM as Registration Service

Whether to use IDM as an external registration service to complete registration for new users. You must configure and enable the IDM Provisioning service to use this option. See "IDM Provisioning".

AM passes IDM these parameters:

- `clientToken`: Signed, encrypted JWT of the OAuth 2.0 authentication state.
- `returnParams`: Encoded URL parameters, required to be returned to AM to resume authentication after registration in IDM is complete.

Default: `False`

amster attribute: `enableRegistrationService`

Create account if it does not exist

When enabled, AM creates an account for the user if the user profile does not exist. If the Prompt for password setting and activation code attribute is enabled, AM prompts the user for a password and activation code before creating the account.

> **Important**
>
> When configured to create new accounts, the SMTP settings must also be valid. As part of account creation, the authentication module sends the resource owner an email with an account activation code. To send the mail, AM uses the SMTP settings you provide in the module configuration.

When disabled, a user without a profile may still log into AM if the Ignore Profile attribute is set in the authentication service of the realm, or if the account is mapped to an anonymous account.

Valid values are:

- `true`
- `false`

Default: `true`

amster attribute: `createAccount`

Account Mapper

Specifies the name of the class that implements the attribute mapping for the account search.

Default: `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper|uid|vkontakte-`

amster attribute: `accountMapperClass`

Account Mapper Configuration

Specifies the attribute configuration used to map the account of the user authenticated in the social provider to the local data store in AM. Valid values take the form *`provider-attr=local-attr`*.

Tip

When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings, by using dot notation.

For example, given a JSON payload of:

```
{
  "sub" : "12345",
  "name" : {
    "first_name" : "Demo",
    "last_name" : "User"
  }
}
```

You can create a mapper such as:

`name.first_name=cn`

Default: `uid=uid`

amster attribute: `accountMapperConfiguration`

Attribute Mapper

Specifies the list of fully qualified class names for implementations that map attributes from the social provider to AM profile attributes.

You can provide a custom attribute mapper. A custom attribute mapper must implement the `org.forgerock.openam.authentication.modules.common.mapping.AttributeMapper` interface.

Provided implementations are:

- `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper`
- `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper` - can only be used when using the `openid` scope

> **Tip**
>
> You can provide string constructor parameters by appending pipe-separated (|) values.
>
> For example, the `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper` class can take two constructor parameters: a comma-separated list of attributes, and a prefix to apply to their values. Specify these as follows:
>
> `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper|uid|vkontakte-`

Default: `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper|uid|vkontakte-`

amster attribute: `attributeMappingClasses`

Attribute Mapper Configuration

Specifies a map of social provider user account attributes to local user profile attributes with values in the form *`provider-attr=local-attr`*.

> **Tip**
>
> When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings, by using dot notation.
>
> For example, given a JSON payload of:
>
> ```
> {
>   "sub" : "12345",
>   "name" : {
>     "first_name" : "Demo",
>     "last_name" : "User"
>   }
> }
> ```
>
> You can create a mapper such as:
>
> `name.first_name=cn`

Default:

```
uid=uid
full_name=givenName
first_name=cn
last_name=sn
```

```
email=mail
```

amster attribute: `attributeMapperConfiguration`

Prompt for password setting and activation code

When enabled, the user must set a password before AM creates an account dynamically. An activation code is also sent to the user's email address. Both the password and the code are required before the account is created.

Valid values are:

- `true`
- `false`

Default: `false`

amster attribute: `promptPasswordFlag`

Map to anonymous user

When enabled, maps the social provider authenticated user to a specified anonymous user. If the Create account if it does not exist property is enabled, AM creates an account for the authenticated user instead of mapping the account to an anonymous user.

Valid values are:

- `true`
- `false`

Default: `false`

amster attribute: `mapToAnonymousUser`

Anonymous User

Specifies an anonymous user that exists in the current realm. The Map to anonymous user property maps authorized users without a profile to this anonyomus user, if enabled.

Default: `anonymous`

amster attribute: `anonymousUserName`

Save attributes in the session

When enabled, saves the values of attributes specified in the Attribute Mapper Configuration property in the AM session.

Valid values are:

- `true`
- `false`

Default: `true`

amster attribute: `saveAttributesInSession`

Email

The following properties are available under the Email tab:

Email attribute in the Response

Specifies the attribute identifying the authenticated user's email address in the response from the profile service in the social provider. This setting is used to send an email message with an activation code for accounts created dynamically.

amster attribute: `emailAttribute`

Mail Server Gateway implementation class

Specifies the class used by the module to send email. A custom subclass of `org.forgerock.openam.authentication.modules.oauth2.EmailGateway` class can be provided.

Default: `org.forgerock.openam.authentication.modules.oauth2.DefaultEmailGatewayImpl`

amster attribute: `emailGateway`

SMTP host

Specifies the host name of the mail server.

Default: `localhost`

amster attribute: `smtpHost`

SMTP port

Specifies the SMTP port number for the mail server.

Default: `25`

amster attribute: `smtpPort`

SMTP User Name

Specifies the username AM uses to authenticate to the mail server.

amster attribute: `smtpUsername`

SMTP User Password

Specifies the password AM uses to authenticate to the mail server.

amster attribute: `smtpPassword`

SMTP SSL Enabled

When enabled, connects to the mail server over SSL. AM must be able to trust the SMTP server certificate.

Valid values are:

- `true`
- `false`

Default: `false`

amster attribute: `smtpSslEnabled`

SMTP From address

Specifies the address of the email sender, such as `no-reply@example.com`.

Default: `info@forgerock.com`

amster attribute: `smtpFromAddress`

Social Authentication Module Properties - WeChat

amster service name: `SocialAuthWeChatModule`

ssoadm service name: `iPlanetAMAuthSocialAuthWeChatService`

Core

The following properties are available under the Core tab:

Authentication Level

Specifies the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

Default: `0`

amster data attribute: `authenticationLevel`

Social Provider

Specifies the name of the social provider for which this module is being set up.

Default: `WeChat`

amster data attribute: `provider`

Client Id

Specifies the `client_id` parameter as described in section 2.2 of The OAuth 2.0 Authorization Framework (RFC 6749).

> **Tip**
>
> To register an application with WeChat and obtain an OAuth 2.0 `client_id` and `client_secret`, visit https://open.weixin.qq.com/cgi-bin/frame?t=home/web_tmpl.

amster attribute: `clientId`

Client Secret

Specifies the `client_secret` parameter as described in section 2.3 of The OAuth 2.0 Authorization Framework (RFC 6749).

amster attribute: `clientSecret`

Authentication Endpoint URL

Specifies the URL to the social provider's endpoint handling authentication as described in section 3.1 of The OAuth 2.0 Authorization Framework (RFC 6749).

Default: `https://open.weixin.qq.com/connect/qrconnect`

amster attribute: `authorizeEndpoint`

Access Token Endpoint URL

Specifies the URL to the endpoint handling access tokens as described in section 3.2 of The OAuth 2.0 Authorization Framework (RFC 6749).

Default: `https://api.wechat.com/sns/oauth2/access_token`

amster attribute: `tokenEndpoint`

User Profile Service URL

Specifies the user profile URL that returns profile information in JSON format.

Default:`https://api.wechat.com/sns/userinfo`

amster attribute: `userInfoEndpoint`

Scope

Specifies a list of user profile attributes that the client application requires, according to *The OAuth 2.0 Authorization Framework (RFC 6749)*. The list depends on the permissions that the resource owner, such as the end user, grants to the client application.

amster attribute: `scope`

Scope Delimiter

Specifies the delimiter used to separate scope values.

Some authorization servers use non-standard separators for scopes. Facebook, for example, uses commas.

Default: space character

amster attribute: `scopeDelimiter`

Subject Property

Specifies the attribute the social provider uses to identify a user.

Default: `openid`

amster attribute: `subjectProperty`

Use Basic Auth

Specifies that the client uses HTTP Basic authentication when authenticating to the social provider.

Valid values are:

- `true`
- `false`

Default: `false`

amster attribute: `usesBasicAuth`

Proxy URL

Specifies the URL to the `/oauth2c/OAuthProxy.jsp` file, which provides AM with GET to POST proxying capabilities. Change this URL only if an external server performs the GET to POST proxying.

Default: `@SERVER_PROTO@://@SERVER_HOST@:@SERVER_PORT@/@SERVER_URI@/oauth2c/OAuthProxy.jsp`

Example: `https://openam.example.com:8443/openam/oauth2c/OAuthProxy.jsp`

amster attribute: `ssoProxyUrl`

Account Provisioning

The following properties are available under the Account Provisioning tab:

Use IDM as Registration Service

Whether to use IDM as an external registration service to complete registration for new users. You must configure and enable the IDM Provisioning service to use this option. See "IDM Provisioning".

AM passes IDM these parameters:

- `clientToken`: Signed, encrypted JWT of the OAuth 2.0 authentication state.
- `returnParams`: Encoded URL parameters, required to be returned to AM to resume authentication after registration in IDM is complete.

Default: `False`

amster attribute: `enableRegistrationService`

Create account if it does not exist

When enabled, AM creates an account for the user if the user profile does not exist. If the Prompt for password setting and activation code attribute is enabled, AM prompts the user for a password and activation code before creating the account.

> **Important**
>
> When configured to create new accounts, the SMTP settings must also be valid. As part of account creation, the authentication module sends the resource owner an email with an account activation code. To send the mail, AM uses the SMTP settings you provide in the module configuration.

When disabled, a user without a profile may still log into AM if the Ignore Profile attribute is set in the authentication service of the realm, or if the account is mapped to an anonymous account.

Valid values are:

- `true`
- `false`

Default: `true`

amster attribute: `createAccount`

Account Provider

Specifies the name of the class that implements the account provider.

Default: `org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider`

amster attribute: `accountProviderClass`

Account Mapper

Specifies the name of the class that implements the attribute mapping for the account search.

Default: `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper|*|wechat-`

amster attribute: `accountMapperClass`

Account Mapper Configuration

Specifies the attribute configuration used to map the account of the user authenticated in the social provider to the local data store in AM. Valid values take the form *`provider-attr=local-attr`*.

Tip

When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings, by using dot notation.

For example, given a JSON payload of:

```
{
  "sub" : "12345",
  "name" : {
    "first_name" : "Demo",
    "last_name" : "User"
  }
}
```

You can create a mapper such as:

`name.first_name=cn`

Default: `openid=uid`

amster attribute: `accountMapperConfiguration`

Attribute Mapper

Specifies the list of fully qualified class names for implementations that map attributes from the social provider to AM profile attributes.

You can provide a custom attribute mapper. A custom attribute mapper must implement the `org.forgerock.openam.authentication.modules.common.mapping.AttributeMapper` interface.

Provided implementations are:

- `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper`
- `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper` - can only be used when using the `openid` scope

> **Tip**
>
> You can provide string constructor parameters by appending pipe-separated (|) values.
>
> For example, the `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper` class can take two constructor parameters: a comma-separated list of attributes, and a prefix to apply to their values. Specify these as follows:
>
> `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper|*|wechat-`

Default: `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper|*|wechat-`

amster attribute: `attributeMappingClasses`

Attribute Mapper Configuration

Specifies a map of social provider user account attributes to local user profile attributes with values in the form *`provider-attr=local-attr`*.

> **Tip**
>
> When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings, by using dot notation.
>
> For example, given a JSON payload of:
>
> ```
> {
>   "sub" : "12345",
>   "name" : {
>     "first_name" : "Demo",
>     "last_name" : "User"
>   }
> }
> ```
>
> You can create a mapper such as:
>
> `name.first_name=cn`

Default:

```
openid=uid
nickname=sn
nickname=cn
nickname=givenName
```

amster attribute: `attributeMapperConfiguration`

Prompt for password setting and activation code

When enabled, the user must set a password before AM creates an account dynamically. An activation code is also sent to the user's email address. Both the password and the code are required before the account is created.

Valid values are:

- `true`
- `false`

Default: `false`

amster attribute: `promptPasswordFlag`

Map to anonymous user

When enabled, maps the social provider authenticated user to a specified anonymous user. If the Create account if it does not exist property is enabled, AM creates an account for the authenticated user instead of mapping the account to an anonymous user.

Valid values are:

- `true`
- `false`

Default: `false`

amster attribute: `mapToAnonymousUser`

Anonymous User

Specifies an anonymous user that exists in the current realm. The Map to anonymous user property maps authorized users without a profile to this anonyomus user, if enabled.

Default: `anonymous`

amster attribute: `anonymousUserName`

Save attributes in the session

When enabled, saves the values of attributes specified in the Attribute Mapper Configuration property in the AM session.

Valid values are:

- `true`
- `false`

Default: `true`

amster attribute: `saveAttributesInSession`

Email

The following properties are available under the Email tab:

Email attribute in the Response

Specifies the attribute identifying the authenticated user's email address in the response from the profile service in the social provider. This setting is used to send an email message with an activation code for accounts created dynamically.

amster attribute: `emailAttribute`

Mail Server Gateway implementation class

Specifies the class used by the module to send email. A custom subclass of `org.forgerock.openam.authentication.modules.oauth2.EmailGateway` class can be provided.

Default: `org.forgerock.openam.authentication.modules.oauth2.DefaultEmailGatewayImpl`

amster attribute: `emailGateway`

SMTP host

Specifies the host name of the mail server.

Default: `localhost`

amster attribute: `smtpHost`

SMTP port

Specifies the SMTP port number for the mail server.

Default: `25`

amster attribute: `smtpPort`

SMTP User Name

Specifies the username AM uses to authenticate to the mail server.

amster attribute: smtpUsername

SMTP User Password

Specifies the password AM uses to authenticate to the mail server.

amster attribute: smtpPassword

SMTP SSL Enabled

When enabled, connects to the mail server over SSL. AM must be able to trust the SMTP server certificate.

Valid values are:

- true
- false

Default: false

amster attribute: smtpSslEnabled

SMTP From address

Specifies the address of the email sender, such as no-reply@example.com.

Default: info@forgerock.com

amster attribute: smtpFromAddress

Social Authentication Module Properties - WeChat Mobile

amster service name: SocialAuthWeChatMobileModule

ssoadm service name: iPlanetAMAuthSocialAuthWeChatMobileService

Core

The following properties are available under the Core tab:

Authentication Level

Specifies the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

Default: 0

amster data attribute: authenticationLevel

Social Provider

Specifies the name of the social provider for which this module is being set up.

Default: `WeChat`

amster data attribute: `provider`

User Profile Service URL

Specifies the user profile URL that returns profile information in JSON format.

Default: `https://api.wechat.com/sns/userinfo`

amster attribute: `userInfoEndpoint`

Scope

Specifies a list of user profile attributes that the client application requires, according to *The OAuth 2.0 Authorization Framework (RFC 6749)*. The list depends on the permissions that the resource owner, such as the end user, grants to the client application.

Default: `snsapi_userinfo`

amster attribute: `scope`

Subject Property

Specifies the attribute the social provider uses to identify a user.

Default: `openid`

amster attribute: `subjectProperty`

Proxy URL

Specifies the URL to the `/oauth2c/OAuthProxy.jsp` file, which provides AM with GET to POST proxying capabilities. Change this URL only if an external server performs the GET to POST proxying.

Default: `@SERVER_PROTO@://@SERVER_HOST@:@SERVER_PORT@/@SERVER_URI@/oauth2c/OAuthProxy.jsp`

Example: `https://openam.example.com:8443/openam/oauth2c/OAuthProxy.jsp`

amster attribute: `ssoProxyUrl`

Account Provisioning

The following properties are available under the Account Provisioning tab:

Use IDM as Registration Service

Whether to use IDM as an external registration service to complete registration for new users. You must configure and enable the IDM Provisioning service to use this option. See "IDM Provisioning".

AM passes IDM these parameters:

- `clientToken`: Signed, encrypted JWT of the OAuth 2.0 authentication state.
- `returnParams`: Encoded URL parameters, required to be returned to AM to resume authentication after registration in IDM is complete.

Default: `False`

amster attribute: `enableRegistrationService`

Create account if it does not exist

When enabled, AM creates an account for the user if the user profile does not exist. If the Prompt for password setting and activation code attribute is enabled, AM prompts the user for a password and activation code before creating the account.

> **Important**
>
> When configured to create new accounts, the SMTP settings must also be valid. As part of account creation, the authentication module sends the resource owner an email with an account activation code. To send the mail, AM uses the SMTP settings you provide in the module configuration.

When disabled, a user without a profile may still log into AM if the Ignore Profile attribute is set in the authentication service of the realm, or if the account is mapped to an anonymous account.

Valid values are:

- `true`
- `false`

Default: `true`

amster attribute: `createAccount`

Account Provider

Specifies the name of the class that implements the account provider.

Default: `org.forgerock.openam.authentication.modules.common.mapping.DefaultAccountProvider`

amster attribute: `accountProviderClass`

Account Mapper

Specifies the name of the class that implements the attribute mapping for the account search.

Default: `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper|*|wechat-`

amster attribute: `accountMapperClass`

Account Mapper Configuration

Specifies the attribute configuration used to map the account of the user authenticated in the social provider to the local data store in AM. Valid values take the form *`provider-attr=local-attr`*.

Tip

When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings, by using dot notation.

For example, given a JSON payload of:

```
{
  "sub" : "12345",
  "name" : {
    "first_name" : "Demo",
    "last_name" : "User"
  }
}
```

You can create a mapper such as:

`name.first_name=cn`

Default: `openid=uid`

amster attribute: `accountMapperConfiguration`

Attribute Mapper

Specifies the list of fully qualified class names for implementations that map attributes from the social provider to AM profile attributes.

You can provide a custom attribute mapper. A custom attribute mapper must implement the `org.forgerock.openam.authentication.modules.common.mapping.AttributeMapper` interface.

Provided implementations are:

- `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper`
- `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper` - can only be used when using the `openid` scope

> **Tip**
>
> You can provide string constructor parameters by appending pipe-separated (|) values.
>
> For example, the `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper` class can take two constructor parameters: a comma-separated list of attributes, and a prefix to apply to their values. Specify these as follows:
>
> `org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper|*|wechat-`

Default: `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper|*|wechat-`

amster attribute: `attributeMappingClasses`

Attribute Mapper Configuration

Specifies a map of social provider user account attributes to local user profile attributes with values in the form *`provider-attr=local-attr`*.

> **Tip**
>
> When using the `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper` class, you can parse JSON objects in mappings, by using dot notation.
>
> For example, given a JSON payload of:
>
> ```
> {
>   "sub" : "12345",
>   "name" : {
>     "first_name" : "Demo",
>     "last_name" : "User"
>   }
> }
> ```
>
> You can create a mapper such as:
>
> `name.first_name=cn`

Default:

```
openid=uid
nickname=sn
nickname=cn
nickname=givenName
```

amster attribute: `attributeMapperConfiguration`

Prompt for password setting and activation code

When enabled, the user must set a password before AM creates an account dynamically. An activation code is also sent to the user's email address. Both the password and the code are required before the account is created.

Valid values are:

- `true`
- `false`

Default: `false`

amster attribute: `promptPasswordFlag`

Map to anonymous user

When enabled, maps the social provider authenticated user to a specified anonymous user. If the Create account if it does not exist property is enabled, AM creates an account for the authenticated user instead of mapping the account to an anonymous user.

Valid values are:

- `true`
- `false`

Default: `false`

amster attribute: `mapToAnonymousUser`

Anonymous User

Specifies an anonymous user that exists in the current realm. The Map to anonymous user property maps authorized users without a profile to this anonyomus user, if enabled.

Default: `anonymous`

amster attribute: `anonymousUserName`

Save attributes in the session

When enabled, saves the values of attributes specified in the Attribute Mapper Configuration property in the AM session.

Valid values are:

- `true`
- `false`

Default: `true`

amster attribute: `saveAttributesInSession`

Email

The following properties are available under the Email tab:

Email attribute in the Response

Specifies the attribute identifying the authenticated user's email address in the response from the profile service in the social provider. This setting is used to send an email message with an activation code for accounts created dynamically.

amster attribute: `emailAttribute`

Mail Server Gateway implementation class

Specifies the class used by the module to send email. A custom subclass of `org.forgerock.openam.authentication.modules.oauth2.EmailGateway` class can be provided.

Default: `org.forgerock.openam.authentication.modules.oauth2.DefaultEmailGatewayImpl`

amster attribute: `emailGateway`

SMTP host

Specifies the host name of the mail server.

Default: `localhost`

amster attribute: `smtpHost`

SMTP port

Specifies the SMTP port number for the mail server.

Default: `25`

amster attribute: `smtpPort`

SMTP User Name

Specifies the username AM uses to authenticate to the mail server.

amster attribute: `smtpUsername`

SMTP User Password

Specifies the password AM uses to authenticate to the mail server.

amster attribute: `smtpPassword`

SMTP SSL Enabled

When enabled, connects to the mail server over SSL. AM must be able to trust the SMTP server certificate.

Valid values are:

- `true`
- `false`

Default: `false`

amster attribute: `smtpSslEnabled`

SMTP From address

Specifies the address of the email sender, such as `no-reply@example.com`.

Default: `info@forgerock.com`

amster attribute: `smtpFromAddress`

Windows Desktop SSO Authentication Module Properties

amster service name: `WindowsDesktopSsoModule`

ssoadm service name: `iPlanetAMAuthWindowsDesktopSSOService`

> **Tip**
>
> Before configuring the authentication module, create an Active Directory account and a `keytab` file.

Service Principal

Specifies the Kerberos principal for authentication in the format `HTTP/`*`host.domain@DC-DOMAIN-NAME`*, where *host.domain* corresponds to the host and domain names of the AM instance and *DC-DOMAIN-NAME* is the domain name of the Kerberos realm (the FQDN of the Active Directory domain). *DC-DOMAIN-NAME* can differ from the domain name for AM.

In multi-server deployments, configure *host.domain* as the load balancer FQDN or IP address in front of the AM instances. For example, `HTTP/openamLB.example.com@KERBEROSREALM.INTERNAL.COM`.

For more information, see the KB article *How do I set up the WDSSO authentication module in AM in a load-balanced environment?*.

amster attribute: `principalName`

ssoadm attribute: `iplanet-am-auth-windowsdesktopsso-principal-name`

Keytab File Name

Specifies the full path of the keytab file for the Service Principal. You generate the keytab file using the Windows **ktpass** utility.

amster attribute: `keytabFileName`

ssoadm attribute: `iplanet-am-auth-windowsdesktopsso-keytab-file`

Kerberos Realm

Specifies the Kerberos Key Distribution Center realm. For the Windows Kerberos service, this is the domain controller server domain name.

amster attribute: `kerberosRealm`

ssoadm attribute: `iplanet-am-auth-windowsdesktopsso-kerberos-realm`

Kerberos Server Name

Specifies the fully qualified domain name of the Kerberos Key Distribution Center server, such as that of the domain controller server.

amster attribute: `kerberosServerName`

ssoadm attribute: `iplanet-am-auth-windowsdesktopsso-kdc`

Return Principal with Domain Name

When enabled, AM automatically returns the Kerberos principal with the domain controller's domain name during authentication.

amster attribute: `returnPrincipalWithDomainName`

ssoadm attribute: `iplanet-am-auth-windowsdesktopsso-returnRealm`

Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

amster attribute: `authenticationLevel`

ssoadm attribute: `iplanet-am-auth-windowsdesktopsso-auth-level`

Trusted Kerberos realms

List of trusted Kerberos realms for user Kerberos tickets. If realms are configured, then Kerberos tickets are only accepted if the realm part of the user principal name of the user's Kerberos ticket matches a realm from the list.

amster attribute: `trustedKerberosRealms`

ssoadm attribute: `iplanet-am-auth-windowsdesktopsso-kerberos-realms-trusted`

isInitiator

Configuration used for the JDK Kerberos LoginModule (`Krb5LoginModule`), which authenticates users using Kerberos principals. Possible values are `true` for initiator credentials, and `false` for acceptor credentials.

Default value: `true`

amster attribute: `kerberosServiceIsinitiator`

ssoadm attribute: `iplanet-am-auth-windowsdesktopsso-kerberos-isinitiator`

Search for the user in the realm

Validates the user against the configured data stores. If the user from the Kerberos token is not found, authentication will fail. If an authentication chain is set, the user is able to authenticate through another module. This search uses the `Alias Search Attribute Name` from the core realm attributes. See "User Profile" for more information about this property.

amster attribute: `lookupUserInRealm`

ssoadm attribute: `iplanet-am-auth-windowsdesktopsso-lookupUserInRealm`

Global Service Properties

The following sections document AM services with configuration properties that affect AM authentication, sessions, and single sign-on:

- "ForgeRock Authenticator (OATH) Service"
- "ForgeRock Authenticator (Push) Service"
- "Push Notification Service"
- "Session"
- "Session Property Whitelist Service"
- "Social Authentication Implementations"
- "WebAuthn Profile Encryption Service"

ForgeRock Authenticator (OATH) Service

amster service name: `AuthenticatorOath`

Realm Defaults

The following settings appear on the **Realm Defaults** tab:

Profile Storage Attribute

Attribute for storing ForgeRock Authenticator OATH profiles.

The default attribute is added to the user store during AM installation. If you want to use a different attribute, you must make sure to add it to your user store schema prior to deploying two-step verification with a ForgeRock OATH authenticator app in AM. AM must be able to write to the attribute.

Default value: `oathDeviceProfiles`

amster attribute: `oathAttrName`

Device Profile Encryption Scheme

Encryption scheme for securing device profiles stored on the server.

If enabled, each device profile is encrypted using a unique random secret key using the given strength of AES encryption in CBC mode with PKCS#5 padding. An HMAC-SHA of the given strength (truncated to half-size) is used to ensure integrity protection and authenticated encryption. The unique random key is encrypted with the given RSA key pair and stored with the device profile.

Note: AES-256 may require installation of the JCE Unlimited Strength policy files.

The possible values for this property are:

- `RSAES_AES256CBC_HS512`. AES-256/HMAC-SHA-512 with RSA Key Wrapping
- `RSAES_AES128CBC_HS256`. AES-128/HMAC-SHA-256 with RSA Key Wrapping
- `NONE`. No encryption of device settings.

Default value: `NONE`

amster attribute: `authenticatorOATHDeviceSettingsEncryptionScheme`

Encryption Key Store

Path to the key store from which to load encryption keys.

Default value: `/path/to/openam/openam/keystore.jks`

amster attribute: `authenticatorOATHDeviceSettingsEncryptionKeystore`

Key Store Type

Type of encryption key store.

Note: PKCS#11 keys tores require hardware support such as a security device or smart card and is not available by default in most JVM installations.

See the JDK 8 PKCS#11 Reference Guide for more details.

The possible values for this property are:

- `JKS`. Java Key Store (JKS).
- `JCEKS`. Java Cryptography Extension Key Store (JCEKS).
- `PKCS11`. PKCS#11 Hardware Crypto Storage.
- `PKCS12`. PKCS#12 Key Store.

Default value: `JKS`

amster attribute: `authenticatorOATHDeviceSettingsEncryptionKeystoreType`

Key Store Password

Password to unlock the key store. This password will be encrypted.

amster attribute: `authenticatorOATHDeviceSettingsEncryptionKeystorePassword`

Key-Pair Alias

Alias of the certificate and private key in the key store. The private key is used to encrypt and decrypt device profiles.

amster attribute: `authenticatorOATHDeviceSettingsEncryptionKeystoreKeyPairAlias`

Private Key Password

Password to unlock the private key.

amster attribute: `authenticatorOATHDeviceSettingsEncryptionKeystorePrivateKeyPassword`

ForgeRock Authenticator (OATH) Device Skippable Attribute Name

The data store attribute that holds the user's decision to enable or disable obtaining and providing a password obtained from the ForgeRock Authenticator app. This attribute must be writeable.

Default value: `oath2faEnabled`

amster attribute: `authenticatorOATHSkippableName`

ForgeRock Authenticator (Push) Service

amster service name: `AuthenticatorPush`

Realm Defaults

The following settings appear on the **Realm Defaults** tab:

Profile Storage Attribute

The user's attribute in which to store Push Notification profiles.

The default attribute is added to the schema when you prepare a user store for use with AM. If you want to use a different attribute, you must make sure to add it to your user store schema prior to deploying push notifications with the ForgeRock Authenticator app in AM. AM must be able to write to the attribute.

Default value: `pushDeviceProfiles`

amster attribute: `pushAttrName`

Device Profile Encryption Scheme

Encryption scheme to use to secure device profiles stored on the server.

If enabled, each device profile is encrypted using a unique random secret key using the given strength of AES encryption in CBC mode with PKCS#5 padding. An HMAC-SHA of the given strength (truncated to half-size) is used to ensure integrity protection and authenticated encryption. The unique random key is encrypted with the given RSA key pair and stored with the device profile.

Note: AES-256 may require installation of the JCE Unlimited Strength policy files.

The possible values for this property are:

- `RSAES_AES256CBC_HS512`. AES-256/HMAC-SHA-512 with RSA Key Wrapping
- `RSAES_AES128CBC_HS256`. AES-128/HMAC-SHA-256 with RSA Key Wrapping
- `NONE`. No encryption of device settings.

Default value: `NONE`

amster attribute: `authenticatorPushDeviceSettingsEncryptionScheme`

Encryption Key Store

Path to the key store from which to load encryption keys.

Default value: `/path/to/openam/openam/keystore.jks`

amster attribute: `authenticatorPushDeviceSettingsEncryptionKeystore`

Key Store Type

Type of key store to load.

Note: PKCS#11 key stores require hardware support such as a security device or smart card and is not available by default in most JVM installations.

See the JDK 8 PKCS#11 Reference Guide for more details.

The possible values for this property are:

- `JKS`. Java Key Store (JKS).
- `JCEKS`. Java Cryptography Extension Key Store (JCEKS).
- `PKCS11`. PKCS#11 Hardware Crypto Storage.
- `PKCS12`. PKCS#12 Key Store.

Default value: `JKS`

amster attribute: `authenticatorPushDeviceSettingsEncryptionKeystoreType`

Key Store Password

Password to unlock the key store. This password is encrypted when it is saved in the AM configuration. You should modify the default value.

amster attribute: `authenticatorPushDeviceSettingsEncryptionKeystorePassword`

Key-Pair Alias

Alias of the certificate and private key in the key store. The private key is used to encrypt and decrypt device profiles.

amster attribute: `authenticatorPushDeviceSettingsEncryptionKeystoreKeyPairAlias`

Private Key Password

Password to unlock the private key.

amster attribute: `authenticatorPushDeviceSettingsEncryptionKeystorePrivateKeyPassword`

ForgeRock Authenticator (Push) Device Skippable Attribute Name

Name of the attribute on a user's profile used to store their selection of whether to skip ForgeRock Authenticator (Push) 2FA modules.

Default value: `push2faEnabled`

amster attribute: `authenticatorPushSkippableName`

IDM Provisioning

amster service name: `IDMProvisioning`

Realm Defaults

The following settings appear on the **Realm Defaults** tab:

Enabled

Default value: `false`

amster attribute: `enabled`

Deployment URL

URL of the IDM deployment, e.g. https://localhost:8080

For example, you might set this property to: *https://openidm.example.com*

amster attribute: `idmDeploymentUrl`

Signing Key Alias

Alias of the signing symmetric key in AM's default keystore. Must be a duplicate of the symmetric key used by IDM.

Default value: `openidm-selfservice-key`

amster attribute: `provisioningSigningKeyAlias`

Encryption Key Alias

Alias of the encryption asymmetric key in AM's default keystore. Must be a duplicate of the asymmetric key used by IDM.

Default value: `selfservice`

amster attribute: `provisioningEncryptionKeyAlias`

Signing Algorithm

JWT signing algorithm.

Default value: `HS256`

amster attribute: `provisioningSigningAlgorithm`

Signing Compatibility Mode

Enable AM to communicate with OpenIDM 6 and earlier.

When this option is enabled, AM will sign JWTs in a way that is compatible with versions of OpenIDM 6 and earlier. The approach used is incompatible with non-extractable HSM keys. Disable this option if you have upgraded to OpenIDM 6.5, or later.

Default value: `false`

amster attribute: `jwtSigningCompatibilityMode`

Encryption Algorithm

JWT encryption algorithm.

Default value: `RSAES_PKCS1_V1_5`

amster attribute: `provisioningEncryptionAlgorithm`

Encryption Method

JWT encryption method.

Default value: `A128CBC_HS256`

amster attribute: `provisioningEncryptionMethod`

Push Notification Service

amster service name: `PushNotification`

Realm Defaults

The following settings appear on the **Realm Defaults** tab:

SNS Access Key ID

Amazon Simple Notification Service Access Key ID. For more information, see Setting up access for Amazon SNS.

For example, you might set this property to: *AKIAIOSFODNN7EXAMPLE*

amster attribute: `accessKey`

SNS Access Key Secret

Amazon Simple Notification Service Access Key Secret. For more information, see Setting up access for Amazon SNS.

For example, you might set this property to: *wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY*

amster attribute: `secret`

SNS Endpoint for APNS

The Simple Notification Service endpoint in Amazon Resource Name format, used to send push messages to the Apple Push Notification Service (APNS).

For example, you might set this property to: *arn:aws:sns:us-east-1:1234567890:app/APNS/production*

amster attribute: `appleEndpoint`

SNS Endpoint for GCM

The Simple Notification Service endpoint in Amazon Resource Name format, used to send push messages over Google Cloud Messaging (GCM).

For example, you might set this property to: *arn:aws:sns:us-east-1:1234567890:app/GCM/production*

amster attribute: `googleEndpoint`

SNS Client Region

Region of your registered Amazon Simple Notification Service client. For more information, see https://docs.aws.amazon.com/general/latest/gr/rande.html.

The possible values for this property are:

- `us-gov-west-1`
- `us-east-1`
- `us-west-1`
- `us-west-2`
- `eu-west-1`
- `eu-central-1`
- `ap-southeast-1`

- `ap-southeast-2`
- `ap-northeast-1`
- `ap-northeast-2`
- `sa-east-1`
- `cn-north-1`

Default value: `us-east-1`

amster attribute: `region`

Message Transport Delegate Factory

The fully qualified class name of the factory responsible for creating the PushNotificationDelegate. The class must implement `org.forgerock.openam.services.push.PushNotificationDelegate`.

Default value: `org.forgerock.openam.services.push.sns.SnsHttpDelegateFactory`

amster attribute: `delegateFactory`

Response Cache Duration

The minimum lifetime to keep unanswered message records in the message dispatcher cache, in seconds. To keep unanswered message records indefinitely, set this property to `0`.Should be tuned so that it is applicable to the use case of this service. For example, the ForgeRock Authenticator (Push) authentication module has a default timeout of 120 seconds.

Default value: `120`

amster attribute: `mdDuration`

Response Cache Concurrency

Level of concurrency to use when accessing the message dispatcher cache. Defaults to `16`, and must be greater than `0`. Choose a value to accommodate as many threads as will ever concurrently access the message dispatcher cache.

Default value: `16`

amster attribute: `mdConcurrency`

Response Cache Size

Maximum size of the message dispatcher cache, in number of records. If set to `0` the cache can grow indefinitely. If the number of records that need to be stored exceeds this maximum, then older items in the cache will be removed to make space.

Default value: `10000`

amster attribute: `mdCacheSize`

Session

amster service name: `session`

Global Attributes

The following settings appear on the **Global Attributes** tab:

Resulting behavior if session quota exhausted

Specify the action to take if a session quota is exhausted:

- **Deny Access**. New session creation requests will be denied.
- **Destroy Next Expiring**. The session that would expire next will be destroyed.
- **Destroy Oldest**. The oldest session will be destroyed.
- **Destroy All**. All previous sessions will be destroyed.

The possible values for this property are:

- `DENY_ACCESS`
- `DESTROY_OLD_SESSION`. DESTROY_OLDEST_SESSION

Default value: `DESTROY_OLD_SESSION`

amster attribute: `behaviourWhenQuotaExhausted`

General

The following settings appear on the **General** tab:

Latest Access Time Update Frequency

Defaults to `60` seconds. At most, AM updates a session's latest access time this often.

Subsequent touches to the session that occur within the specified number of seconds after an update will not cause additional updates to the session's access time.

Refreshing a session returns the idle time as the number of seconds since an update has occurred, which will be between `0` and the specified Latest Access Time Update Frequency.

Default value: `60`

amster attribute: `latestAccessTimeUpdateFrequency`

DN Restriction Only Enabled

If enabled, AM will not perform DNS lookups when checking restrictions in cookie hijacking mode.

Default value: `false`

amster attribute: `dnRestrictionOnly`

Session Timeout Handler implementations

Lists plugin classes implementing session timeout handlers. Specify the fully qualified name.

amster attribute: `timeoutHandlers`

Session Search

The following settings appear on the **Session Search** tab:

Maximum Number of Search Results

Maximum number of results from a session search. Do not set this attribute to a large value, for example more than 1000, unless sufficient system resources are allocated.

Default value: `120`

amster attribute: `maxSessionListSize`

Timeout for Search

Time after which AM sees an incomplete search as having failed, in seconds.

Default value: `5`

amster attribute: `sessionListRetrievalTimeout`

Session Property Change Notifications

The following settings appear on the **Session Property Change Notifications** tab:

Enable Property Change Notifications

If on, then AM notifies other applications participating in SSO when a session property in the Notification Properties list changes on a CTS-based session.

The possible values for this property are:

- `ON`
- `OFF`

Default value: `OFF`

amster attribute: `propertyChangeNotifications`

Notification Properties

Lists session properties for which AM can send notifications upon modification. Session notification applies to CTS-based sessions only.

amster attribute: `notificationPropertyList`

Session Quotas

The following settings appear on the **Session Quotas** tab:

Enable Quota Constraints

If on, then AM allows you to set quota constraints on CTS-based sessions.

The possible values for this property are:

- `ON`
- `OFF`

Default value: `OFF`

amster attribute: `iplanet-am-session-enable-session-constraint`

Read Timeout for Quota Constraint

Maximum wait time after which AM considers a search for live session count as having failed if quota constraints are enabled, in milliseconds.

Default value: `6000`

amster attribute: `quotaConstraintMaxWaitTime`

Resulting behavior if session quota exhausted

Specify the action to take if a session quota is exhausted:

- **Deny Access**. New session creation requests will be denied.

- **Destroy Next Expiring**. The session that would expire next will be destroyed.
- **Destroy Oldest**. The oldest session will be destroyed.
- **Destroy All**. All previous sessions will be destroyed.

The possible values for this property are:

- `org.forgerock.openam.session.service.DenyAccessAction`. Deny Access
- `org.forgerock.openam.session.service.DestroyNextExpiringAction`. Destroy Next Expiring
- `org.forgerock.openam.session.service.DestroyOldestAction`. Destroy Oldest
- `org.forgerock.openam.session.service.DestroyAllAction`. Destroy All

Default value: `org.forgerock.openam.session.service.DestroyNextExpiringAction`

amster attribute: `behaviourWhenQuotaExhausted`

Deny user login when session repository is down

This property only takes effect when the session quota constraint is enabled, and the session data store is unavailable.

The possible values for this property are:

- `YES`
- `NO`

Default value: `NO`

amster attribute: `denyLoginWhenRepoDown`

Client-based Sessions

The following settings appear on the **Client-based Sessions** tab:

Signing Algorithm Type

Specifies the algorithm that AM uses to sign the JSON Web Token (JWT) containing the session content. Signing the JWT enables tampering detection.

AM supports the following signing algorithms:

- **HS256**. HMAC using SHA-256.
- **HS384**. HMAC using SHA-384.

- **HS512**. HMAC using SHA-512.
- **RS256**. RSASSA-PKCS1-v1_5 using SHA-256.
- **ES256**. ECDSA using SHA-256 and NIST standard P-256 elliptic curve.
- **ES384**. ECDSA using SHA-384 and NIST standard P-384 elliptic curve.
- **ES512**. ECDSA using SHA-512 and NIST standard P-521 elliptic curve.

The possible values for this property are:

- `NONE`
- `HS256`
- `HS384`
- `HS512`
- `RS256`
- `ES256`
- `ES384`
- `ES512`

Default value: `HS256`

amster attribute: `statelessSigningType`

Signing HMAC Shared Secret

Specifies the shared secret that AM uses when performing HMAC signing on the session JWT.

Specify a shared secret when using a "Signing Algorithm Type" of `HS256`, `HS384`, or `HS512`.

amster attribute: `statelessSigningHmacSecret`

Signing RSA/ECDSA Certificate Alias

Specify the alias of a certificate containing a public/private key pair that AM uses when performing RSA or ECDSA signing on the session JWT. Specify a signing certificate alias when using a "Signing Algorithm Type" of `RS256`, `ES256`, `ES384`, or `ES512`.

The certificate is retrieved from the keystore specified by the `com.sun.identity.saml.xmlsig.keystore` property.

Default value: `test`

amster attribute: `statelessSigningRsaCertAlias`

Encryption Algorithm

Specifies the algorithm that AM uses to encrypt the JSON Web Token (JWT) containing the session content.

AM supports the following encryption algorithms:

- **NONE**. No encryption is selected.
- **RSA**. Session content is encrypted with AES using a unique key. The key is then encrypted with an RSA public key and appended to the JWT.

  AM supports the following padding modes, which you can set using the `org.forgerock.openam.session.stateless.rsa.padding` advanced property:

  - `RSA1_5`. RSA with PKCS#1 v1.5 padding.
  - `RSA-OAEP`. RSA with optimal asymmetric encryption padding (OAEP) and SHA-1.
  - `RSA-OAEP-256`. RSA with OAEP padding and SHA-256.
- **AES KeyWrapping**. Session content is encrypted with AES using a unique key and is then wrapped using AES KeyWrap and the master key. This provides additional security, compared to RSA, at the cost of 128 or 256 bits (or 32 bytes) depending on the size of the master key. This method provides authenticated encryption, which removes the need for a separate signature and decreases the byte size of the JWT. See RFC 3394.
- **Direct AES Encryption**. Session content is encrypted with direct AES encryption with a symmetric key. This method provides authenticated encryption, which removes the need for a separate signature and decreases the byte size of the JWT.

Important: To prevent users from accidentally disabling all authentication support, which can be accomplished by disabling signing and not using an authenticated encryption mode, you must set the `org.forgerock.openam.session.stateless.signing.allownone` system property to `true` to turn off signing completely.

The possible values for this property are:

- `NONE`
- `RSA`
- `AES_KEYWRAP`. AES KeyWrapping
- `DIRECT`. Direct AES encryption

Default value: `DIRECT`

amster attribute: `statelessEncryptionType`

Encryption RSA Certificate Alias

Specifies the alias of a certificate containing a public/private key pair that AM uses when encrypting a JWT. Specify an encryption certificate alias when using an Encryption Algorithm Type of `RSA`.

The certificate is retrieved from the keystore referenced by the `com.sun.identity.saml.xmlsig.keystore` property.

Default value: `test`

amster attribute: `statelessEncryptionRsaCertAlias`

Encryption Symmetric AES Key

AES key for use with Direct or AES KeyWrap encryption modes.

The symmetric AES key is a base64-encoded random key.

For direct encryption with `AES-GCM` or for `AES-KeyWrap` with any content encryption method, this should be 128, 192, or 256 bits.

For direct encryption with `AES-CBC-HMAC`, the key should be double those sizes (one half for the AES key, the other have for the HMAC key).

AES key sizes greater than 128 bits require installation of the JCE Unlimited Strength policy files in your JRE.

amster attribute: `statelessEncryptionAesKey`

Compression Algorithm

If enabled the session state is compressed before signing and encryption.

WARNING: Enabling compression may compromise encryption. This may leak information about the content of the session state if encryption is enabled.

The possible values for this property are:

- `NONE`
- `DEF`. Deflate Compression.

Default value: `NONE`

amster attribute: `statelessCompressionType`

Enable Session Blacklisting

Blacklists client-based sessions that log out.

We recommend enabling this setting if the maximum session time is high. Blacklist state is stored in the Core Token Service (CTS) token store until the session expires, in order to ensure that sessions cannot continue to be used. Requires a server restart for changes to take effect.

Default value: `false`

amster attribute: `openam-session-stateless-enable-session-blacklisting`

Session Blacklist Cache Size

Number of blacklisted sessions to cache in memory to speed up blacklist checks and reduce load on the CTS. The cache size should be approximately the number of logouts expected in the maximum session time.

Default value: `10000`

amster attribute: `openam-session-stateless-blacklist-cache-size`

Blacklist Poll Interval (seconds)

Specifies the interval at which AM polls the Core Token Service to update the list of logged out sessions, in seconds.

The longer the polling interval, the more time a malicious user has to connect to other AM servers in a deployment and make use of a stolen session cookie. Shortening the polling interval improves the security for logged out sessions, but might incur a minimal decrease in overall AM performance due to increased network activity. Set to `0` to disable this feature completely.

Default value: `60`

amster attribute: `openam-session-stateless-blacklist-poll-interval`

Blacklist Purge Delay (minutes)

When added to the maximum session time, specifies the amount of time that AM tracks logged out sessions.

Increase the blacklist purge delay if you expect system clock skews in a deployment of AM servers to be greater than one minute. There is no need to increase the blacklist purge delay for servers running a clock synchronization protocol, such as Network Time Protocol.

Default value: `1`

amster attribute: `openam-session-stateless-blacklist-purge-delay`

Dynamic Attributes

> **Note**
>
> Configuring any of the following properties at the realm level (Realms > *Realm Name* > Services > Session) causes the values to be stored in the identity data store configured in that realm.
>
> If you remove the identity data store from the realm, the properties will use the values configured at the global level (Configure > Global Services > Session).

The following settings appear on the **Dynamic Attributes** tab:

Maximum Session Time

Maximum time a session can remain valid before AM requires the user to authenticate again, in minutes.

Default value: `120`

amster attribute: `maxSessionTime`

Maximum Idle Time

Maximum time a CTS-based session can remain idle before AM requires the user to authenticate again, in minutes.

Default value: `30`

amster attribute: `maxIdleTime`

Maximum Caching Time

Maximum time that external clients of AM are recommended to cache the session for, in minutes.

Default value: `3`

amster attribute: `maxCachingTime`

Active User Sessions

Maximum number of concurrent CTS-based sessions AM allows a user to have.

Default value: `5`

amster attribute: `quotaLimit`

Session Property Whitelist Service

amster service name: `SessionPropertyWhiteList`

Realm Defaults

The following settings appear on the **Realm Defaults** tab:

Whitelisted Session Property Names

A list of properties that users may read, edit the value of, or delete from their session.

Adding properties to sessions can impact AM's performance. Because there is no size constraint limiting the set of properties that you can add to sessions, and no limit on the number of session properties you can add, keep in mind that adding session properties can increase the load on an AM deployment in the following areas:

- AM server memory
- DS storage
- DS replication

Protected attributes will NOT be allowed to be set, edited or deleted, even if they are included in this whitelist.

Default value: `AMCtxId`

amster attribute: `sessionPropertyWhitelist`

Important

Use of this property to retrieve user attributes from a session using REST only applies to authentication modules in pre-AM 6.5.3 versions, and to authentication trees and modules in AM 6.5.3 or later.

Social Authentication Implementations

amster service name: `SocialAuthentication`

Realm Defaults

The following settings appear on the **Realm Defaults** tab:

Display Names

The display names for the implementations - this will be used to provide a name for the icon displayed on the login page. The key should be used across all the settings on this page to join them together.

For example:

Key	Value
google	Google

amster attribute: `displayNames`

Authentication Chains

The name of the authentication chains that are the entry points to being authenticated by each respective social authentication provider. The key should correspond to a key used to define a Display Name above.

For example:

Key	Value
google	socialAuthChainGoogle

amster attribute: `authenticationChains`

Icons

Either a full URL or a path relative to the base of the site/server where the image can be found. The image will be used on the login page to link to the authentication chain defined above. The key should correspond to a key used to define a Display Name above.

For example:

Key	Value
google	/images/google-sign-in.png

amster attribute: `icons`

Enabled Implementations

Provide a key that has been used to define the settings above to enable that set of settings.

For example: google

amster attribute: `enabledKeys`

WebAuthn Profile Encryption Service

amster service name: `AuthenticatorWebAuthn`

Realm Defaults

The following settings appear on the **Realm Defaults** tab:

Profile Storage Attribute

The user's attribute in which to store WebAuthn profiles.

The default attribute is added to the schema when you prepare a user store for use with AM. If you want to use a different attribute, you must make sure to add it to your user store schema prior to deploying webauthn with AM. AM must be able to write to the attribute.

Default value: `webauthnDeviceProfiles`

amster attribute: `webauthnAttrName`

Device Profile Encryption Scheme

Encryption scheme to use to secure device profiles stored on the server.

If enabled, each device profile is encrypted using a unique random secret key using the given strength of AES encryption in CBC mode with PKCS#5 padding. An HMAC-SHA of the given strength (truncated to half-size) is used to ensure integrity protection and authenticated encryption. The unique random key is encrypted with the given RSA key pair and stored with the device profile.

Note: AES-256 may require installation of the JCE Unlimited Strength policy files.

The possible values for this property are:

- `RSAES_AES256CBC_HS512`. AES-256/HMAC-SHA-512 with RSA Key Wrapping
- `RSAES_AES128CBC_HS256`. AES-128/HMAC-SHA-256 with RSA Key Wrapping
- `NONE`. No encryption of device settings.

Default value: `NONE`

amster attribute: `authenticatorWebAuthnDeviceSettingsEncryptionScheme`

Encryption Key Store

Path to the key store from which to load encryption keys.

Default value: `/path/to/openam/openam/keystore.jceks`

amster attribute: `authenticatorWebAuthnDeviceSettingsEncryptionKeystore`

Key Store Type

Type of key store to load.

Note: PKCS#11 key stores require hardware support such as a security device or smart card and is not available by default in most JVM installations.

See the JDK 8 PKCS#11 Reference Guide for more details.

The possible values for this property are:

- `JKS`. Java Key Store (JKS).
- `JCEKS`. Java Cryptography Extension Key Store (JCEKS).
- `PKCS11`. PKCS#11 Hardware Crypto Storage.
- `PKCS12`. PKCS#12 Key Store.

Default value: `JCEKS`

amster attribute: `authenticatorWebAuthnDeviceSettingsEncryptionKeystoreType`

Key Store Password

Password to unlock the key store. This password is encrypted when it is saved in the AM configuration. You should modify the default value.

amster attribute: `authenticatorWebAuthnDeviceSettingsEncryptionKeystorePassword`

Key-Pair Alias

Alias of the certificate and private key in the key store. The private key is used to encrypt and decrypt device profiles.

amster attribute: `authenticatorWebAuthnDeviceSettingsEncryptionKeystoreKeyPairAlias`

Private Key Password

Password to unlock the private key.

amster attribute: `authenticatorWebAuthnDeviceSettingsEncryptionKeystorePrivateKeyPassword`

Authentication API Functionality

This section covers the available functionality when Scripting authentication modules use client-side and server-side authentication script types.

Tip

When developing server-side scripts, it can be useful to increase the debug level of the `org.apache.http.wire` and `org.apache.http.headers` appenders to `Message`.

By default, these appenders are always set to the Warning level unless logging is disabled. For more information, see the org.forgerock.allow.http.client.debug advanced server property.

Authentication API functionality includes:

- "Accessing Authentication State"
- "Accessing Profile Data"
- "Accessing Client-Side Script Output Data"
- "Accessing Request Data"
- "Redirecting the User After Authentication Failure"

Accessing Authentication State

AM passes authState and sharedState objects to server-side scripts in order for the scripts to access authentication state.

Server-side scripts can access the current authentication state through the authState object.

The authState value is SUCCESS if the authentication is currently successful, or FAILED if authentication has failed. Server-side scripts must set a value for authState before completing.

If an earlier authentication module in the authentication chain has set the login name of the user, server-side scripts can access the login name through username.

The following authentication modules set the login name of the user:

- Anonymous
- Certificate
- Data Store
- Federation
- HTTP Basic
- JDBC
- LDAP
- Membership
- RADIUS
- SecurID
- Windows Desktop SSO

Accessing Profile Data

Server-side authentication scripts can access profile data through the methods of the `idRepository` object.

Profile Data Methods

Method	Parameters	Return Type	Description
`idRepository.getAttribute`	*`User Name`* (type: `String`) *`Attribute Name`* (type: `String`)	`Set`	Return the values of the named attribute for the named user.
`idRepository.setAttribute`	*`User Name`* (type: `String`) *`Attribute Name`* (type: `String`) *`Attribute Values`* (type: `Array`)	`Void`	Set the named attribute as specified by the attribute value for the named user, and persist the result in the user's profile.
`idRepository.addAttribute`	*`User Name`* (type: `String`) *`Attribute Name`* (type: `String`) *`Attribute Value`* (type: `String`)	`Void`	Add an attribute value to the list of attribute values associated with the attribute name for a particular user.

Accessing Client-Side Script Output Data

Client-side scripts add data they gather into a string object named `clientScriptOutputData`. Client-side scripts then cause the user-agent automatically to return the data to AM by HTTP POST of a self-submitting form.

Accessing Request Data

Server-side scripts can get access to the login request by using the methods of the `requestData` object.

The following table lists the methods of the `requestData` object. Note that this object differs from the client-side `requestData` object and contains information about the original authentication request made by the user.

Request Data Methods

Method	Parameters	Return Type	Description
`requestData.getHeader`	*`Header Name`* (type: `String`)	`String`	Return the String value of the named request header, or `null` if parameter is not set.
`requestData.getHeaders`	*`Header Name`* (type: `String`)	`String[]`	Return the array of String values of the named request header, or `null` if parameter is not set.

Method	Parameters	Return Type	Description
`requestData.getParameter`	*`Parameter Name`* (type: `String`)	`String`	Return the String value of the named request parameter, or `null` if parameter is not set.
`requestData.getParameters`	*`Parameter Name`* (type: `String`)	`String[]`	Return the array of String values of the named request parameter, or `null` if parameter is not set.

Redirecting the User After Authentication Failure

Server-side scripts can redirect the user to a specific URL in case of authentication failure by adding a `gotoOnFailureUrl` property to the chain's shared state.

When the script reaches a `FAILED` authentication state (defined by the `authState` variable), it checks if the `gotoOnFailureUrl` property is stored in the shared state. If so, the script redirects the user to the specified URL.

You can redirect the user to a page relative to AM's URL, or to an absolute URL:

Relative URL

```
...
sharedState.put("gotoOnFailureUrl","/openam/XUI/?service=testChain#failedLogin");
authState = FAILED;
...
```

Absolute URL

```
...
sharedState.put("gotoOnFailureUrl","http://www.example.com");
authState = FAILED;
...
```

Note that the failure URL relative to AM's domain includes the authentication service; this is so that when the user clicks on the link to log in again, AM constructs the login page with the appropriate service instead of with the default one for the realm.

When redirecting the user to an absolute URL different from AM's scheme, FQDN, and port, you must configure the URL in the Validation Service of the realm. Otherwise, AM will ignore the redirection. For more information, see "To Configure the Validation Service".

Scripted Decision Node API Functionality

This section covers the functionality provided by the Decision node script for authentication trees script type.

Scripted Decision Node API functionality includes:

- Accessing Request Header Data

- Accessing Shared State Data
- Encrypting and Decrypting Shared State Data
- Accessing Existing Session Data
- Using Callbacks

The node also has access to the functionality provided in the global scripting API, for example:

- Accessing HTTP Services
- Debug Logging

Accessing Request Header Data

Scripted Decision Node scripts can access the headers provided by the login request by using the methods of the `requestHeaders` object.

Note that the script has access to a copy of the headers. Changing their values does not affect the request itself.

The following table lists the methods of the `requestHeaders` object:

Request Headers Methods

Method	**Parameters**	**Return Type**	**Description**

`requestHeaders.get`	*`Header Name`* (type: `String`)	`String[]`	Return the array of string values of the named request header, or `null` if the property is not set. Note that header names are case-sensitive.
			For example:
			`var headerName = "user-agent";`
			
			`if`
			` (requestHeaders.get(headerName).get(0).indexOf("Chrome") !`
			`== -1) {`
			`    outcome = "true";`
			`} else {`
			`    outcome = "false";`
			`}`

Accessing Shared State Data

Scripted Decision Node scripts can get access to the shared state within the tree by using the `sharedState` and `transientState` objects.

The following table lists the available methods:

Shared State Methods

Method	Parameters	Return Type	Description

`sharedState.get`	*Property Name* (type: `String`)	`String`	Return the string value of the named shared state property, or `null` if the property is not set. Note that property names are case-sensitive.
			For example, use the following code to get the current authentication level:
			`var currentAuthLevel =`
			` sharedState.get("authLevel");`

`transientState.get`	*Property Name* (type: `String`)	`String`	Return the string value of the named transient state property, or `null` if the property is not set. Note that property names are case-sensitive. For example, use the following code to get the value of a previously supplied password: `var givenPassword = transientState.get("password");`

Encrypting and Decrypting Shared State Data

(Added in AM 6.5.4) Scripted Decision Node scripts can encrypt and decrypt data from the shared state using the `sharedStateCrypto` object.

> **Important**
>
> Use of the `sharedStateCrypto` object is intended to encrypt and decrypt one-time passwords along with their timestamps only.
>
> The `org.forgerock.openam.auth.nodes.crypto.NodeSharedStateCrypto` Java class included in AM 6.5.4 or later and its `sharedStateCrypto` scripting binding do not exist in AM 7 or later because `org.forgerock.openam.auth.nodes.crypto.NodeSharedStateCrypto` no longer exists.
>
> Any scripts or authentication nodes using this class and/or binding will need to be updated accordingly when upgrading to AM 7 or later because `org.forgerock.openam.auth.nodes.crypto.NodeSharedStateCrypto` no longer exists.

The following table lists the available methods:

Encrypting and Decrypting Methods

Method	Parameters	Return Type	Description
`sharedStateCrypto decrypt`	*Property Name* (type: `String`)	`JSON`	Decrypts a string value and returns a JSON, or `null` if the string payload is not set. If something is missing or is misconfigured in terms of the secret, a secret-

Method	Parameters	Return Type	Description

			related exception is thrown. Note that property names are case-sensitive.
			For example, use the following code to decrypt the string containing the one-time password, and to assign the decrypted password's to a variable:
			`var otpEncrypted =`
			`sharedState.get("oneTimePasswordEncrypted");`
			`var otpJsonValue =`
			`sharedStateCrypto.decrypt(otpEncrypted);`
			`var otp =`
			`otpJsonValue.get("oneTimePassword").asString();`

`sharedStateCrypto`
`encrypt`
	`Property Name` (type: `JSON`)	`String`	Encrypts a JSON object.
			For example, use the following code to encrypt the JSON containing the one-time password:
			`var otpJson =`
			`org.forgerock.json.JsonValue.json(org.forgerock.json.JsonVal`
			`org.forgerock.json.JsonValue.field("oneTimePassword",`
			`"19219283"),`
			`org.forgerock.json.JsonValue.field("oneTimePasswordTimestamp`
			`1623328298)`
			`));`
			`var otpEncrypted =`
			`sharedStateCrypto.encrypt(otpJson);`
			`sharedState.put("oneTimePasswordEncrypted",`
			`otpEncrypted);`

Accessing Existing Session Data

Scripted Decision Node scripts can access any existing session information provided during a session upgrade request by using the `existingSession` object.

The following table lists the methods of the `existingSession` object:

Existing Session Methods

Method	Parameters	Return Type	Description
`existingSession.get`	*`Property Name`* (type: `String`)	`String`	Return the string value of the named existing session property, or `null` if the property is not set. Note that property names are case-sensitive.

Method	Parameters	Return Type	Description

			Warning
			If the current request is not a session upgrade and does not provide an existing session, the `existingSession` variable is not declared. Check for a declaration before attempting to access the variable.
			For example, use the following code to get the authentication level of the existing session:
			`if (typeof existingSession !== 'undefined')`
			`{`
			`  existingAuthLevel = existingSession.get("AuthLevel");`
			`}`
			`else`
			`{`
			`  logger.error("Variable existingSession not declared - not a session upgrade.");`
			`}`

Using Callbacks

The scripted decision node can use callbacks to provide or request additional information during the authentication process.

For example, the following scripts use the `NameCallBack` callback to request a "Nickname" value from the user, and adds the returned value to the `sharedState` map for use elsewhere in the authentication tree:

Groovy

```
import org.forgerock.openam.auth.node.api.*;
import javax.security.auth.callback.NameCallback;

if (callbacks.isEmpty()) {
  action = Action.send(new NameCallback("Enter Your Nickname")).build();
} else {
  sharedState = sharedState.put("Nickname", callbacks.get(0).getName());
  action = Action.goTo("true").build();
}
```

JavaScript

```
var fr = JavaImporter(
  org.forgerock.openam.auth.node.api,
  javax.security.auth.callback.NameCallback
);
with (fr) {
  if (callbacks.isEmpty()) {
    action = Action.send(new NameCallback("Enter Your Nickname")).build();
  } else {
    sharedState = sharedState.put("Nickname", callbacks.get(0).getName());
    action = Action.goTo("true").build();
  }
}
```

For a list of supported callbacks, see "Supported Callbacks".

OAuth 2.0 Access Token Modification Scripting API

The following properties are available to scripts:

`clientProperties`

A map of properties configured in the relevant client profile. Only present if the client was correctly identified.

The keys in the map are as follows:

`clientId`

The URI of the client.

`allowedGrantTypes`

The list of the allowed grant types (`org.forgerock.oauth2.core.GrantType`) for the client.

`allowedResponseTypes`

The list of the allowed response types for the client.

`allowedScopes`

The list of the allowed scopes for the client.

`customProperties`

A map of any custom properties added to the client.

Lists or maps are included as sub-maps. For example, a custom property of `customMap[Key1]=Value1` is returned as `customMap` > `Key1` > `Value1`.

To add custom properties to a client, go to OAuth 2.0 > Clients > *Client ID* > Advanced, and then update the Custom Properties field.

`requestProperties`

A map of the properties present in the request. Always present.

The keys in the map are as follows:

`requestUri`

The URI of the request.

`realm`

The realm to which the request was made.

`requestParams`

The request parameters, and/or posted data. Each value in this map is a list of one, or more, properties.

> **Important**
>
> To mitigate the risk of reflection type attacks, use OWASP best practices when handling these properties. For example, see Unsafe use of Reflection.

`scopes`

Contains a set of the requested scopes. For example:

```
[
    "profile",
    "friends"
]
```

`scriptName`

The display name of the script. Always present.

Appendix A. About the REST API

This appendix shows how to use the RESTful interfaces for direct integration between web client applications and ForgeRock Access Management.

Introducing REST

Representational State Transfer (REST) is an architectural style that sets certain constraints for designing and building large-scale distributed hypermedia systems.

As an architectural style, REST has very broad applications. The designs of both HTTP 1.1 and URIs follow RESTful principles. The World Wide Web is no doubt the largest and best known REST application. Many other web services also follow the REST architectural style. Examples include OAuth 2.0, OpenID Connect 1.0, and User-Managed Access (UMA).

The ForgeRock Common REST (CREST) API applies RESTful principles to define common verbs for HTTP-based APIs that access web resources and collections of web resources.

Interface Stability: Evolving

Most native AM REST APIs use the CREST verbs. (In contrast, OAuth 2.0, OpenID Connect 1.0 and UMA APIs follow their respective standards.)

About ForgeRock Common REST

ForgeRock® Common REST is a common REST API framework. It works across the ForgeRock platform to provide common ways to access web resources and collections of resources. Adapt the examples in this section to your resources and deployment.

Note

This section describes the full Common REST framework. Some platform component products do not implement all Common REST behaviors exactly as described in this section. For details, refer to the product-specific examples and reference information in other sections of this documentation set.

Common REST Resources

Servers generally return JSON-format resources, though resource formats can depend on the implementation.

Resources in collections can be found by their unique identifiers (IDs). IDs are exposed in the resource URIs. For example, if a server has a user collection under `/users`, then you can access a user at `/users/user-id`. The ID is also the value of the `_id` field of the resource.

Resources are versioned using revision numbers. A revision is specified in the resource's `_rev` field. Revisions make it possible to figure out whether to apply changes without resource locking and without distributed transactions.

Common REST Verbs

The Common REST APIs use the following verbs, sometimes referred to collectively as CRUDPAQ. For details and HTTP-based examples of each, follow the links to the sections for each verb.

Create

Add a new resource.

This verb maps to HTTP PUT or HTTP POST.

For details, see "Create".

Read

Retrieve a single resource.

This verb maps to HTTP GET.

For details, see "Read".

Update

Replace an existing resource.

This verb maps to HTTP PUT.

For details, see "Update".

Delete

Remove an existing resource.

This verb maps to HTTP DELETE.

For details, see "Delete".

Patch

Modify part of an existing resource.

This verb maps to HTTP PATCH.

For details, see "Patch".

Action

Perform a predefined action.

This verb maps to HTTP POST.

For details, see "Action".

Query

Search a collection of resources.

This verb maps to HTTP GET.

For details, see "Query".

Common REST Parameters

Common REST reserved query string parameter names start with an underscore, `_`.

Reserved query string parameters include, but are not limited to, the following names:

`_action`
`_api`
`_crestapi`
`_fields`
`_mimeType`
`_pageSize`
`_pagedResultsCookie`
`_pagedResultsOffset`
`_prettyPrint`
`_queryExpression`

`_queryFilter`
`_queryId`
`_sortKeys`
`_totalPagedResultsPolicy`

> **Note**
>
> Some parameter values are not safe for URLs, so URL-encode parameter values as necessary.

Continue reading for details about how to use each parameter.

Common REST Extension Points

The *action* verb is the main vehicle for extensions. For example, to create a new user with HTTP POST rather than HTTP PUT, you might use `/users?_action=create`. A server can define additional actions. For example, `/tasks/1?_action=cancel`.

A server can define *stored queries* to call by ID. For example, `/groups?_queryId=hasDeletedMembers`. Stored queries can call for additional parameters. The parameters are also passed in the query string. Which parameters are valid depends on the stored query.

Common REST API Documentation

Common REST APIs often depend at least in part on runtime configuration. Many Common REST endpoints therefore serve *API descriptors* at runtime. An API descriptor documents the actual API as it is configured.

Use the following query string parameters to retrieve API descriptors:

`_api`

Serves an API descriptor that complies with the OpenAPI specification.

This API descriptor represents the API accessible over HTTP. It is suitable for use with popular tools such as Swagger UI.

`_crestapi`

Serves a native Common REST API descriptor.

This API descriptor provides a compact representation that is not dependent on the transport protocol. It requires a client that understands Common REST, as it omits many Common REST defaults.

> **Note**
>
> Consider limiting access to API descriptors in production environments in order to avoid unnecessary traffic.

> To provide documentation in production environments, see "To Publish OpenAPI Documentation" instead.

To Publish OpenAPI Documentation

In production systems, developers expect stable, well-documented APIs. Rather than retrieving API descriptors at runtime through Common REST, prepare final versions, and publish them alongside the software in production.

Use the OpenAPI-compliant descriptors to provide API reference documentation for your developers as described in the following steps:

1. Configure the software to produce production-ready APIs.

   In other words, the software should be configured as in production so that the APIs are identical to what developers see in production.

2. Retrieve the OpenAPI-compliant descriptor.

   The following command saves the descriptor to a file, `myapi.json`:

   ```
   $ curl -o myapi.json endpoint?_api
   ```

3. (Optional) If necessary, edit the descriptor.

   For example, you might want to add security definitions to describe how the API is protected.

   If you make any changes, then also consider using a source control system to manage your versions of the API descriptor.

4. Publish the descriptor using a tool such as Swagger UI.

   You can customize Swagger UI for your organization as described in the documentation for the tool.

Create

There are two ways to create a resource, either with an HTTP POST or with an HTTP PUT.

To create a resource using POST, perform an HTTP POST with the query string parameter `_action=create` and the JSON resource as a payload. Accept a JSON response. The server creates the identifier if not specified:

```
POST /users?_action=create HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
{ JSON resource }
```

To create a resource using PUT, perform an HTTP PUT including the case-sensitive identifier for the resource in the URL path, and the JSON resource as a payload. Use the `If-None-Match: *` header. Accept a JSON response:

```
PUT /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
If-None-Match: *
{ JSON resource }
```

The `_id` and content of the resource depend on the server implementation. The server is not required to use the `_id` that the client provides. The server response to the create request indicates the resource location as the value of the `Location` header.

If you include the `If-None-Match` header, its value must be `*`. In this case, the request creates the object if it does not exist, and fails if the object does exist. If you include the `If-None-Match` header with any value other than `*`, the server returns an HTTP 400 Bad Request error. For example, creating an object with `If-None-Match: revision` returns a bad request error. If you do not include `If-None-Match: *`, the request creates the object if it does not exist, and *updates* the object if it does exist.

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

Read

To retrieve a single resource, perform an HTTP GET on the resource by its case-sensitive identifier (`_id`) and accept a JSON response:

```
GET /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
```

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_fields=`*`field`*`[,`*`field`*`...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

`_mimeType=`*`mime-type`*

Some resources have fields whose values are multi-media resources such as a profile photo for example.

By specifying both a single *field* and also the *mime-type* for the response content, you can read a single field value that is a multi-media resource.

In this case, the content type of the field value returned matches the *mime-type* that you specify, and the body of the response is the multi-media resource.

The `Accept` header is not used in this case. For example, `Accept: image/png` does not work. Use the `_mimeType` query string parameter instead.

Update

To update a resource, perform an HTTP PUT including the case-sensitive identifier (`_id`) as the final element of the path to the resource, and the JSON resource as the payload. Use the `If-Match: `*`_rev`* header to check that you are actually updating the version you modified. Use `If-Match: *` if the version does not matter. Accept a JSON response:

```
PUT /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
If-Match: _rev
{ JSON resource }
```

When updating a resource, include all the attributes to be retained. Omitting an attribute in the resource amounts to deleting the attribute unless it is not under the control of your application. Attributes not under the control of your application include private and read-only attributes. In addition, virtual attributes and relationship references might not be under the control of your application.

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

Delete

To delete a single resource, perform an HTTP DELETE by its case-sensitive identifier (`_id`) and accept a JSON response:

```
DELETE /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
```

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

Patch

To patch a resource, send an HTTP PATCH request with the following parameters:

- `operation`
- `field`
- `value`
- `from` (optional with copy and move operations)

You can include these parameters in the payload for a PATCH request, or in a JSON PATCH file. If successful, you'll see a JSON response similar to:

```
PATCH /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
If-Match: _rev
{ JSON array of patch operations }
```

PATCH operations apply to three types of targets:

- **single-valued**, such as an object, string, boolean, or number.
- **list semantics array**, where the elements are ordered, and duplicates are allowed.
- **set semantics array**, where the elements are not ordered, and duplicates are not allowed.

ForgeRock PATCH supports several different `operations`. The following sections show each of these operations, along with options for the `field` and `value`:

Patch Operation: Add

The `add` operation ensures that the target field contains the value provided, creating parent fields as necessary.

If the target field is single-valued, then the value you include in the PATCH replaces the value of the target. Examples of a single-valued field include: object, string, boolean, or number.

An add operation has different results on two standard types of arrays:

- **List semantic arrays**: you can run any of these add operations on that type of array:
  - If you add an array of values, the PATCH operation appends it to the existing list of values.
  - If you add a single value, specify an ordinal element in the target array, or use the {-} special index to add that value to the end of the list.
- **Set semantic arrays**: The value included in the patch is merged with the existing set of values. Any duplicates within the array are removed.

As an example, start with the following list semantic array resource:

```
{
    "fruits" : [ "orange", "apple" ]
}
```

The following add operation includes the pineapple to the end of the list of fruits, as indicated by the - at the end of the fruits array.

```
{
    "operation" : "add",
    "field" : "/fruits/-",
    "value" : "pineapple"
}
```

The following is the resulting resource:

```
{
    "fruits" : [ "orange", "apple", "pineapple" ]
}
```

Note that you can add only one array element one at a time, as per the corresponding JSON Patch specification. If you add an array of elements, for example:

```
{
    "operation" : "add",
    "field" : "/fruits/-",
    "value" : ["pineapple", "mango"]
}
```

The resulting resource would have the following invalid JSON structure:

```
{
    "fruits" : [ "orange", "apple", ["pineapple", "mango"]]
}
```

Patch Operation: Copy

The copy operation takes one or more existing values from the source field. It then adds those same values on the target field. Once the values are known, it is equivalent to performing an add operation on the target field.

The following copy operation takes the value from a field named mail, and then runs a replace operation on the target field, another_mail.

```
[
  {
     "operation":"copy",
     "from":"mail",
     "field":"another_mail"
  }
]
```

If the source field value and the target field value are configured as arrays, the result depends on whether the array has list semantics or set semantics, as described in "Patch Operation: Add".

Patch Operation: Increment

The increment operation changes the value or values of the target field by the amount you specify. The value that you include must be one number, and may be positive or negative. The value of the target field must accept numbers. The following increment operation adds 1000 to the target value of /user/payment.

```
[
  {
    "operation" : "increment",
    "field" : "/user/payment",
    "value" : "1000"
  }
]
```

Since the value of the increment is a single number, arrays do not apply.

Patch Operation: Move

The move operation removes existing values on the source field. It then adds those same values on the target field. It is equivalent to performing a remove operation on the source, followed by an add operation with the same values, on the target.

The following move operation is equivalent to a remove operation on the source field, surname, followed by a replace operation on the target field value, lastName. If the target field does not exist, it is created.

```
[
  {
     "operation":"move",
     "from":"surname",
     "field":"lastName"
  }
]
```

To apply a move operation on an array, you need a compatible single-value, list semantic array, or set semantic array on both the source and the target. For details, see the criteria described in "Patch Operation: Add".

Patch Operation: Remove

The `remove` operation ensures that the target field no longer contains the value provided. If the remove operation does not include a value, the operation removes the field. The following `remove` deletes the value of the `phoneNumber`, along with the field.

```
[
  {
    "operation" : "remove",
    "field" : "phoneNumber"
  }
]
```

If the object has more than one `phoneNumber`, those values are stored as an array.

A `remove` operation has different results on two standard types of arrays:

- **List semantic arrays**: A `remove` operation deletes the specified element in the array. For example, the following operation removes the first phone number, based on its array index (zero-based):

  ```
  [
     {
        "operation" : "remove",
        "field" : "/phoneNumber/0"
     }
  ]
  ```

- **Set semantic arrays**: The list of values included in a patch are removed from the existing array.

Patch Operation: Replace

The `replace` operation removes any existing value(s) of the targeted field, and replaces them with the provided value(s). It is essentially equivalent to a `remove` followed by a `add` operation. If the arrays are used, the criteria is based on "Patch Operation: Add". However, indexed updates are not allowed, even when the target is an array.

The following `replace` operation removes the existing `telephoneNumber` value for the user, and then adds the new value of `+1 408 555 9999`.

```
[
  {
    "operation" : "replace",
    "field" : "/telephoneNumber",
    "value" : "+1 408 555 9999"
  }
]
```

A PATCH replace operation on a list semantic array works in the same fashion as a PATCH remove operation. The following example demonstrates how the effect of both operations. Start with the following resource:

```
{
    "fruits" : [ "apple", "orange", "kiwi", "lime" ],
}
```

Apply the following operations on that resource:

```
[
  {
    "operation" : "remove",
    "field" : "/fruits/0",
    "value" : ""
  },
  {
    "operation" : "replace",
    "field" : "/fruits/1",
    "value" : "pineapple"
  }
]
```

The PATCH operations are applied sequentially. The `remove` operation removes the first member of that resource, based on its array index, (`fruits/0`), with the following result:

```
[
  {
    "fruits" : [ "orange", "kiwi", "lime" ],
  }
]
```

The second PATCH operation, a `replace`, is applied on the second member (`fruits/1`) of the intermediate resource, with the following result:

```
[
  {
    "fruits" : [ "orange", "pineapple", "lime" ],
  }
]
```

Patch Operation: Transform

The `transform` operation changes the value of a field based on a script or some other data transformation command. The following `transform` operation takes the value from the field named `/objects`, and applies the `something.js` script as shown:

```
[
  {
    "operation" : "transform",
    "field" : "/objects",
    "value" : {
      "script" : {
        "type" : "text/javascript",
        "file" : "something.js"
      }
    }
  }
]
```

Patch Operation Limitations

Some HTTP client libraries do not support the HTTP PATCH operation. Make sure that the library you use supports HTTP PATCH before using this REST operation.

For example, the Java Development Kit HTTP client does not support PATCH as a valid HTTP method. Instead, the method `HttpURLConnection.setRequestMethod("PATCH")` throws `ProtocolException`.

Parameters

You can use the following parameters. Other parameters might depend on the specific action implementation:

`_prettyPrint=true`

Format the body of the response.

`_fields=`*`field`*`[,`*`field`*`...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

Action

Actions are a means of extending Common REST APIs and are defined by the resource provider, so the actions you can use depend on the implementation.

The standard action indicated by `_action=create` is described in "Create".

Parameters

You can use the following parameters. Other parameters might depend on the specific action implementation:

`_prettyPrint=true`

Format the body of the response.

`_fields=`*`field`*`[,`*`field`*`...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

Query

To query a resource collection (or resource container if you prefer to think of it that way), perform an HTTP GET and accept a JSON response, including at least a `_queryExpression`, `_queryFilter`, or `_queryId` parameter. These parameters cannot be used together:

```
GET /users?_queryFilter=true HTTP/1.1
Host: example.com
Accept: application/json
```

The server returns the result as a JSON object including a "results" array and other fields related to the query string parameters that you specify.

Parameters

You can use the following parameters:

`_queryFilter=`*`filter-expression`*

Query filters request that the server return entries that match the filter expression. You must URL-escape the filter expression.

The string representation is summarized as follows. Continue reading for additional explanation:

```
Expr           = OrExpr
OrExpr         = AndExpr ( 'or' AndExpr ) *
AndExpr        = NotExpr ( 'and' NotExpr ) *
NotExpr        = '!' PrimaryExpr | PrimaryExpr
PrimaryExpr    = '(' Expr ')' | ComparisonExpr | PresenceExpr | LiteralExpr
ComparisonExpr = Pointer OpName JsonValue
PresenceExpr   = Pointer 'pr'
LiteralExpr    = 'true' | 'false'
Pointer        = JSON pointer
OpName         = 'eq' |  # equal to
                 'co' |  # contains
                 'sw' |  # starts with
                 'lt' |  # less than
                 'le' |  # less than or equal to
                 'gt' |  # greater than
                 'ge' |  # greater than or equal to
                 STRING  # extended operator
JsonValue      = NUMBER | BOOLEAN | '"' UTF8STRING '"'
STRING         = ASCII string not containing white-space
UTF8STRING     = UTF-8 string possibly containing white-space
```

JsonValue components of filter expressions follow RFC 7159: *The JavaScript Object Notation (JSON) Data Interchange Format*. In particular, as described in section 7 of the RFC, the escape character in strings is the backslash character. For example, to match the identifier `test\`, use `_id eq 'test\\'`. In the JSON resource, the `\` is escaped the same way: `"_id":"test\\"`.

When using a query filter in a URL, be aware that the filter expression is part of a query string parameter. A query string parameter must be URL encoded as described in RFC 3986: *Uniform Resource Identifier (URI): Generic Syntax* For example, white space, double quotes (`"`), parentheses, and exclamation characters need URL encoding in HTTP query strings. The following rules apply to URL query components:

```
query       = *( pchar / "/" / "?" )
pchar       = unreserved / pct-encoded / sub-delims / ":" / "@"
unreserved  = ALPHA / DIGIT / "-" / "." / "_" / "~"
pct-encoded = "%" HEXDIG HEXDIG
sub-delims  = "!" / "$" / "&" / "'" / "(" / ")"
                  / "*" / "+" / "," / ";" / "="
```

`ALPHA`, `DIGIT`, and `HEXDIG` are core rules of RFC 5234: *Augmented BNF for Syntax Specifications*:

```
ALPHA       =  %x41-5A / %x61-7A   ; A-Z / a-z
DIGIT       =  %x30-39             ; 0-9
HEXDIG      =  DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
```

As a result, a backslash escape character in a *JsonValue* component is percent-encoded in the URL query string parameter as `%5C`. To encode the query filter expression `_id eq 'test\\'`, use `_id+eq+'test%5C%5C'`, for example.

A simple filter expression can represent a comparison, presence, or a literal value.

For comparison expressions use *json-pointer comparator json-value*, where the *comparator* is one of the following:

`eq` (equals)
`co` (contains)
`sw` (starts with)
`lt` (less than)
`le` (less than or equal to)
`gt` (greater than)
`ge` (greater than or equal to)

For presence, use *json-pointer pr* to match resources where:

- The JSON pointer is present.

- The value it points to is not `null`.

Literal values include true (match anything) and false (match nothing).

Complex expressions employ `and`, `or`, and `!` (not), with parentheses, `(expression)`, to group expressions.

`_queryId=`*`identifier`*

Specify a query by its identifier.

Specific queries can take their own query string parameter arguments, which depend on the implementation.

`_pagedResultsCookie=string`

The string is an opaque cookie used by the server to keep track of the position in the search results. The server returns the cookie in the JSON response as the value of `pagedResultsCookie`.

In the request `_pageSize` must also be set and non-zero. You receive the cookie value from the provider on the first request, and then supply the cookie value in subsequent requests until the server returns a `null` cookie, meaning that the final page of results has been returned.

The `_pagedResultsCookie` parameter is supported when used with the `_queryFilter` parameter. The `_pagedResultsCookie` parameter is not guaranteed to work when used with the `_queryExpression` and `_queryId` parameters.

The `_pagedResultsCookie` and `_pagedResultsOffset` parameters are mutually exclusive, and not to be used together.

`_pagedResultsOffset=integer`

When `_pageSize` is non-zero, use this as an index in the result set indicating the first page to return.

The `_pagedResultsCookie` and `_pagedResultsOffset` parameters are mutually exclusive, and not to be used together.

`_pageSize=integer`

Return query results in pages of this size. After the initial request, use `_pagedResultsCookie` or `_pageResultsOffset` to page through the results.

`_totalPagedResultsPolicy=string`

When a `_pageSize` is specified, and non-zero, the server calculates the "totalPagedResults", in accordance with the `totalPagedResultsPolicy`, and provides the value as part of the response. The "totalPagedResults" is either an estimate of the total number of paged results (`_totalPagedResultsPolicy=ESTIMATE`), or the exact total result count (`_totalPagedResultsPolicy=EXACT`). If no count policy is specified in the query, or if `_totalPagedResultsPolicy=NONE`, result counting is disabled, and the server returns value of -1 for "totalPagedResults".

`_sortKeys=[+-]field[,[+-]field...]`

Sort the resources returned based on the specified field(s), either in `+` (ascending, default) order, or in `-` (descending) order.

Because ascending order is the default, including the `+` character in the query is unnecessary. If you do include the `+`, it must be URL-encoded as `%2B`, for example:

```
http://localhost:8080/api/users?_prettyPrint=true&_queryFilter=true&_sortKeys=%2Bname/givenName
```

The `_sortKeys` parameter is not supported for predefined queries (`_queryId`).

`_prettyPrint=true`

Format the body of the response.

`_fields=`*`field`*`[,`*`field...`*`]`

Return only the specified fields in each element of the "results" array in the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

HTTP Status Codes

When working with a Common REST API over HTTP, client applications should expect at least the following HTTP status codes. Not all servers necessarily return all status codes identified here:

200 OK

The request was successful and a resource returned, depending on the request.

201 Created

The request succeeded and the resource was created.

204 No Content

The action request succeeded, and there was no content to return.

304 Not Modified

The read request included an `If-None-Match` header, and the value of the header matched the revision value of the resource.

400 Bad Request

The request was malformed.

401 Unauthorized

The request requires user authentication.

403 Forbidden

Access was forbidden during an operation on a resource.

404 Not Found

The specified resource could not be found, perhaps because it does not exist.

405 Method Not Allowed

The HTTP method is not allowed for the requested resource.

406 Not Acceptable

The request contains parameters that are not acceptable, such as a resource or protocol version that is not available.

409 Conflict

The request would have resulted in a conflict with the current state of the resource.

410 Gone

The requested resource is no longer available, and will not become available again. This can happen when resources expire for example.

412 Precondition Failed

The resource's current version does not match the version provided.

415 Unsupported Media Type

The request is in a format not supported by the requested resource for the requested method.

428 Precondition Required

The resource requires a version, but no version was supplied in the request.

500 Internal Server Error

The server encountered an unexpected condition that prevented it from fulfilling the request.

501 Not Implemented

The resource does not support the functionality required to fulfill the request.

503 Service Unavailable

The requested resource was temporarily unavailable. The service may have been disabled, for example.

Cross-Site Request Forgery (CSRF) Protection

AM includes a global filter to harden AM's protection against CSRF attacks. The filter applies to all REST endpoints under `json/` and requires that all requests other than GET, HEAD, or OPTIONS include, at least, one of the following headers:

- `X-Requested-With`

  This header is often sent by Javascript frameworks, and the XUI already sends it on all requests.

- `Accept-API-Version`

  This header specifies which version of the REST API to use. Use this header in your requests to ensure future changes to the API do not affect your clients.

  For more information about API versioning, see "REST API Versioning".

Failure to include at least one of the headers would cause the REST call to fail with a `403 Forbidden` error, even if the SSO token is valid.

To disable the filter, navigate to Configure > Global Services > REST APIs > and turn off Enable CSRF Protection.

The `json/` endpoint is not vulnerable to CSRF attacks when the filter is disabled, since it requires the `"Content-Type: application/json"` header, which currently triggers the same protection in browsers. This may change in the future, so it is recommended to enable the CSRF filter.

REST API Versioning

In OpenAM 12.0.0 and later, REST API features are assigned version numbers.

Providing version numbers in the REST API helps ensure compatibility between releases. The version number of a feature increases when AM introduces a non-backwards-compatible change that affects clients making use of the feature.

AM provides versions for the following aspects of the REST API.

resource

> Any changes to the structure or syntax of a returned response will incur a *resource* version change. For example changing `errorMessage` to `message` in a JSON response.

protocol

> Any changes to the methods used to make REST API calls will incur a *protocol* version change. For example changing `_action` to `$action` in the required parameters of an API feature.

To ensure your clients are always compatible with a newer version of AM, you should always include resource versions in your REST calls.

Supported REST API Versions

For information about the supported protocol and resource versions available in AM, see the API Explorer in the *Development Guide* available in the AM console.

The *AM Release Notes* section, "*Changes and Deprecated Functionality*" in the *Release Notes* describes the differences between API versions.

Specifying an Explicit REST API Version

You can specify which version of the REST API to use by adding an `Accept-API-Version` header to the request. The following example requests *resource* version 2.0 and *protocol* version 1.0:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: demo" \
--header "X-OpenAM-Password: changeit" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
```

You can configure the default behavior AM will take when a REST call does not specify explicit version information. For more information, see "Configuring the Default REST API Version for a Deployment".

Configuring the Default REST API Version for a Deployment

You can configure the default behavior AM will take when a REST call does not specify explicit version information using either of the following procedures:

- "Configure Versioning Behavior by using the AM Console"
- "Configure Versioning Behavior by Using the ssoadm Command"

The available options for default behavior are as follows:

Latest

The latest available supported version of the API is used.

This is the preset default for new installations of AM.

Oldest

The oldest available supported version of the API is used.

This is the preset default for upgraded AM instances.

> **Note**
>
> The oldest supported version may not be the first that was released, as APIs versions become deprecated or unsupported. See "Deprecated Functionality" in the *Release Notes*.

None

No version will be used. When a REST client application calls a REST API without specifying the version, AM returns an error and the request fails.

Configure Versioning Behavior by using the AM Console

1. Log in as AM administrator, `amadmin`.

2. Click Configure > Global Services, and then click REST APIs.

3. In Default Version, select the required response to a REST API request that does not specify an explicit version: `Latest`, `Oldest`, or `None`.

4. (Optional) Optionally, enable `Warning Header` to include warning messages in the headers of responses to requests.

5. Save your work.

Configure Versioning Behavior by Using the ssoadm Command

- Use the **ssoadm set-attr-defs** command with the `openam-rest-apis-default-version` attribute set to either `Latest`, `Oldest` or `None`, as in the following example:

```
$ ssh openam.example.com
$ cd /path/to/openam-tools/admin/openam/bin
$ ./ssoadm \
 set-attr-defs \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --servicename RestApisService \
 --schematype Global \
 --attributevalues openam-rest-apis-default-version=None
Schema attribute defaults were set.
```

REST API Versioning Messages

AM provides REST API version messages in the JSON response to a REST API call. You can also configure AM to return version messages in the response headers.

Messages include:

- Details of the REST API versions used to service a REST API call.

- Warning messages if REST API version information is not specified or is incorrect in a REST API call.

The `resource` and `protocol` version used to service a REST API call are returned in the `Content-API-Version` header, as shown below:

```
$ curl \
-i \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: demo" \
--header "X-OpenAM-Password: changeit" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
HTTP/1.1 200 OK
Content-API-Version: protocol=1.0,resource=2.0
Server: Restlet-Framework/2.1.7
Content-Type: application/json;charset=UTF-8

{
    "tokenId":"AQIC5wM...TU3OQ*",
    "successUrl":"/openam/console"
}
```

If the default REST API version behavior is set to `None`, and a REST API call does not include the `Accept-API-Version` header, or does not specify a `resource` version, then a `400 Bad Request` status code is returned, as shown below:

```
$ curl \
--header "Content-Type: application/json" \
--header "Accept-API-Version: protocol=1.0" \
https://openam.example.com:8443/openam/json/realms/root/serverinfo/*
{
    "code":400,
    "reason":"Bad Request",
    "message":"No requested version specified and behavior set to NONE."
}
```

If a REST API call does include the `Accept-API-Version` header, but the specified `resource` or `protocol` version does not exist in AM, then a `404 Not Found` status code is returned, as shown below:

```
$ curl \
--header "Content-Type: application/json" \
--header "Accept-API-Version: protocol=1.0, resource=999.0" \
https://openam.example.com:8443/openam/json/realms/root/serverinfo/*
{
    "code":404,
    "reason":"Not Found",
    "message":"Accept-API-Version: Requested version \"999.0\" does not match any routes."
}
```

Tip

For more information on setting the default REST API version behavior, see "Specifying an Explicit REST API Version".

Specifying Realms in REST API Calls

This section describes how to work with realms when making REST API calls to AM.

Realms can be specified in the following ways when making a REST API call to AM:

DNS Alias

When making a REST API call, the DNS alias of a realm can be specified in the subdomain and domain name components of the REST endpoint.

To list all users in the top-level realm use the DNS alias of the AM instance, for example, the REST endpoint would be:

```
https://openam.example.com:8443/openam/json/users?_queryId=*
```

To list all users in a realm with DNS alias `suppliers.example.com` the REST endpoint would be:

```
https://suppliers.example.com:8443/openam/json/users?_queryId=*
```

Path

When making a REST API call, specify the realm in the path component of the endpoint. You must specify the entire hierarchy of the realm, starting at the top-level realm. Prefix each realm in the hierarchy with the `realms/` keyword. For example `/realms/root/realms/customers/realms/europe`.

To authenticate a user in the top-level realm, use the `root` keyword. For example:

```
https://openam.example.com:8443/openam/json/realms/root/authenticate
```

To authenticate a user in a subrealm named `customers` within the top-level realm, the REST endpoint would be:

```
https://openam.example.com:8443/openam/json/realms/root/realms/customers/authenticate
```

If realms are specified using both the DNS alias and path methods, the path is used to determine the realm.

For example, the following REST endpoint returns users in a subrealm of the top-level realm named `europe`, not the realm with DNS alias `suppliers.example.com`:

```
https://suppliers.example.com:8443/openam/json/realms/root/realms/europe/users?_queryId=*
```

Authentication and Logout using REST

You can use REST-like APIs under `/json/authenticate` and `/json/sessions` for authentication and for logout.

The `/json/authenticate` endpoint does not support the CRUDPAQ verbs and therefore does not technically satisfy REST architectural requirements. The term *REST-like* describes this endpoint better than *REST*.

After a successful authentication, AM returns a `tokenId` object that applications can present as a cookie value for other operations that require authentication. This object is a session token—a representation of the exchange of information and credentials between AM and the user or identity.

The type of `tokenId` returned varies depending on where AM stores the sessions for the realm to which the user authenticates:

- If CTS-based sessions are enabled, the `tokenId` object is a reference to the session state stored in the CTS token store.
- If client-based sessions are enabled, the `tokenId` object is the session state for that particular user or identity.

Developers should be aware that the size of the `tokenId` for client-based sessions—2000 bytes or greater—is considerably longer than for CTS-based sessions—approximately 100 bytes. For more information about session tokens, see "Session Cookies".

Authenticating to AM using REST

To log in to AM using REST, make an HTTP POST request to the `json/authenticate` endpoint. You must specify the entire hierarchy of the realm, starting at the Top Level Realm. Prefix each realm in the hierarchy with the `realms/` keyword. For example, `/realms/root/realms/customers/realms/europe`.

AM uses the default authentication service configured for the realm. You can override the default by specifying authentication services and other options in the REST request.

AM provides both simple authentication methods, such as providing user name and password, and complex authentication journeys that may involve a tree with inner tree evaluation and/or multi-factor authentication.

For authentication journeys where providing a user name and password is enough, you can log in to AM using a **curl** command similar to the following:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: demo" \
--header "X-OpenAM-Password: changeit" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
{
    "tokenId": "AQIC5w...NTcy*",
    "successUrl": "/openam/console",
    "realm":"/"
}
```

The user name and password are sent in headers. This zero page login mechanism works only for name/password authentication.

Note that the POST body is empty; otherwise, AM interprets the body as a continuation of an existing authentication attempt, one that uses a supported callback mechanism. AM implements callback mechanisms to support complex authentication journeys, such as those where the user needs to be redirected to a third party or interact with a device as part of multi-factor authentication.

When a client makes a call to the `/json/authenticate` endpoint appending a valid SSO token, AM returns the `tokenId` field **empty** when `HttpOnly` cookies are enabled. For example:

```
{
    "tokenId":"",
    "successUrl":"/openam/console",
    "realm":"/"
}
```

Tip

About Success and Failure URLs

On authentication success, AM returns an SSO token and a success URL. By default, users are redirected to `/openam/console`.

No failure URL is configured by default. When configured, on authentication failure, AM returns HTTP status code 401 Unauthorized and the failure URL:

```
{
    "code":401,
    "reason":"Unauthorized",
    "message":"Login failure",
    "failureUrl": "http://www.example.com/401.html"
}
```

For more information about configuring successful or failed authentication, see "Configuring Success and Failure Redirection URLs".

Using UTF-8 User Names

To use UTF-8 user names and passwords in calls to the `/json/authenticate` endpoint, base64-encode the string, and then wrap the string as described in RFC 2047:

```
encoded-word = "=?" charset "?" encoding "?" encoded-text "?="
```

For example, to authenticate using a UTF-8 username, such as `dëmø`, perform the following steps:

1. Encode the string in base64 format: `yZfDq8mxw7g=`.

2. Wrap the base64-encoded string as per RFC 2047: `=?UTF-8?B?yZfDq8mxw7g=?=`.

3. Use the result in the `X-OpenAM-Username` header passed to the authentication endpoint as follows:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: =?UTF-8?B?yZfDq8mxw7g=?=" \
--header "X-OpenAM-Password: changeit" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
      {
    "tokenId": "AQIC5w...NTcy*",
    "successUrl": "/openam/console",
    "realm":"/"
}
```

Authenticating to Specific Authentication Services

You can provide AM with additional information about how you are authenticating. For example, you can specify the authentication tree you want to use, or request from AM a list of the authentication services that would satisfy a particular authentication condition.

The following example shows how to specify the `ldapService` chain by using the `authIndexType` and `authIndexValue` query string parameters:

```
$ curl \
--request POST \
--header "X-OpenAM-Username: demo" \
--header "X-OpenAM-Password: changeit" \
--header 'Accept-API-Version: resource=2.0, protocol=1.0' \
'https://openam.example.com:8443/openam/json/realms/root/authenticate?
authIndexType=service&authIndexValue=ldapService'
```

You can exchange the `ldapService` chain with any other chain or tree.

For more information about using the `authIndexType` parameter to authenticate to specific services, see "Authenticate Endpoint Parameters".

Authenticate Endpoint Parameters

To authenticate to AM using REST, make an HTTP POST request to the `/json/authenticate` endpoint. You must specify the entire hierarchy of the realm, starting at the Top Level Realm. Prefix each realm in the hierarchy with the `realms/` keyword. For example, `/realms/root/realms/customers/realms/europe`.

The following list describes the `/json/authenticate` endpoint supported parameters:

`authIndexType`

> Specifies the type of authentication the user will perform. Always use in conjunction with the `authIndexValue` parameter to provide additional information about the way the user is authenticating.
>
> If not specified, AM authenticates the user against the default authentication service configured for the realm.
>
> The `authIndexType` parameter supports the following types:
>
> - composite_advice
>
>   Specifies that the value of the `authIndexValue` parameter is a URL-encoded composite advice string.
>
>   Use `composite_advice` when you want to give AM hints of which authentication services to use when logging in a user. For example, use an authentication service that provides an authentication level of 10 or higher:

```
$ curl -get \
--request POST \
--header "Content-Type: application/json" \
--header 'Accept-API-Version: resource=2.0, protocol=1.0' \
--data-urlencode 'authIndexType=composite_advice' \
--data-urlencode 'authIndexValue=<Advices>
    <AttributeValuePair>
        <Attribute name="AuthLevelConditionAdvice"/>
        <Value>10</Value>
    </AttributeValuePair>
</Advices>' \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
```

Note that the previous `curl` command URL-encodes the XML values, and the -G parameter appends them as query string parameters to the URL.

Possible options for advices are:

- `TransactionConditionAdvice`. Requires the unique ID of a transaction token. For example:

  ```
  <Advices>
    <AttributeValuePair>
      <Attribute name="TransactionConditionAdvice"/>
      <Value>9dae2c80-fe7a-4a36-b57b-4fb1271b0687</Value>
    </AttributeValuePair>
  </Advices>
  ```

  For more information, see "*Implementing Transactional Authorization*" in the *Authorization Guide*.

- `AuthenticateToServiceConditionAdvice`. Requires the name of an authentication chain or tree. For example:

  ```
  <Advices>
    <AttributeValuePair>
      <Attribute name="AuthenticateToServiceConditionAdvice"/>
      <Value>myExampleTree</Value>
    </AttributeValuePair>
  </Advices>
  ```

- `AuthSchemeConditionAdvice`. Requires the name of an authentication module. For example:

  ```
  <Advices>
    <AttributeValuePair>
      <Attribute name="AuthSchemeConditionAdvice"/>
      <Value>DataStoreModule</Value>
    </AttributeValuePair>
  </Advices>
  ```

- `AuthenticateToRealmConditionAdvice`. Requires the name of a realm. For example:

```
<Advices>
  <AttributeValuePair>
    <Attribute name="AuthenticateToRealmConditionAdvice"/>
    <Value>myRealm</Value>
  </AttributeValuePair>
</Advices>
```

- `AuthLevelConditionAdvice`. Requires an authentication level. For example:

```
<Advices>
  <AttributeValuePair>
    <Attribute name="AuthLevelConditionAdvice"/>
    <Value>10</Value>
  </AttributeValuePair>
</Advices>
```

- `AuthenticateToTreeConditionAdvice`. Requires the name of an authentication tree. For example:

```
<Advices>
  <AttributeValuePair>
    <Attribute name="AuthenticateToTreeConditionAdvice"/>
    <Value>PersistentCookieTree</Value>
  </AttributeValuePair>
</Advices>
```

You can specify multiple advice conditions and combine them. For example:

```
<Advices>
  <AttributeValuePair>
    <Attribute name="AuthenticateToServiceConditionAdvice"/>
    <Value>ldapService</Value>
  </AttributeValuePair>
  <AttributeValuePair>
    <Attribute name="AuthenticateToServiceConditionAdvice"/>
    <Value>Example</Value>
  </AttributeValuePair>
  <AttributeValuePair>
    <Attribute name="AuthLevelConditionAdvice"/>
    <Value>10</Value>
  </AttributeValuePair>
</Advices>
```

- level

  Specifies that the value of the `authIndexValue` parameter is the minimum authentication level an authentication service must satisfy to log in the user.

  For example, to log into AM using an authentication service that provides a minimum authentication level of 10, you could use the following:

```
$ curl \
--request POST \
--header 'Accept-API-Version: resource=2.0, protocol=1.0' \
'https://openam.example.com:8443/openam/json/realms/root/authenticate?
authIndexType=level&authIndexValue=10'
```

- module

  Specifies that the value of the `authIndexValue` parameter is the name of the authentication module AM must use to log in the user.

  For example, to log into AM using the built-in `DataStore` authentication module, you could use the following:

  ```
  $ curl \
  --request POST \
  --header 'Accept-API-Version: resource=2.0, protocol=1.0' \
  'https://openam.example.com:8443/openam/json/realms/root/authenticate?
  authIndexType=module&authIndexValue=DataStore'
  ```

- resource

  Specifies that the value of the `authIndexValue` parameter is a URL protected by an AM policy.

  For example, to log into AM using a policy matching the `http://www.example.com` resource, you could use the following:

  ```
  $ curl \
  --request POST \
  --header 'Accept-API-Version: resource=2.0, protocol=1.0' \
  'https://openam.example.com:8443/openam/json/realms/root/authenticate?
  authIndexType=resource&authIndexValue=http%3A%2F%2Fwww.example.com'
  ```

  Note that the resource must be URL-encoded. Authentication will fail if no policy matches the resource.

- service

  Specifies that the value of the `authIndexValue` parameter is the name of an authentication tree or authentication chain AM must use to log in the user.

  For example, to log in to AM using the built-in `ldapService` authentication chain, you could use the following:

  ```
  $ curl \
  --request POST \
  --header 'Accept-API-Version: resource=2.0, protocol=1.0' \
  'https://openam.example.com:8443/openam/json/realms/root/authenticate?
  authIndexType=service&authIndexValue=ldapService'
  ```

- user

  Specifies that the value of the `authIndexValue` parameter is a valid user ID. AM will then authenticate the user against the chain configured in the User Authentication Configuration field of that user's profile.

  For example, for the user `demo` to log into AM using the chain specified in their user profile, you could use the following:

```
$ curl \
--request POST \
--header 'Accept-API-Version: resource=2.0, protocol=1.0' \
'https://openam.example.com:8443/openam/json/realms/root/authenticate?
authIndexType=user&authIndexValue=demo'
```

Authentication will fail if the User Authentication Configuration field is empty for the user.

If several authentication services that satisfy the authentication requirements are available, AM presents them as a choice callback to the user. Return the required callbacks to AM to authenticate.

Required: No.

`authIndexValue`

Specifies the value of the `authIndexType` parameter.

Required: Yes, when using the `authIndexType` parameter.

`noSession`

When set to `true`, specifies that AM should not return a session when authenticating a user. For example:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--header "X-OpenAM-Username: demo" \
--header "X-OpenAM-Password: changeit" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate?noSession=true'
{
    "message":"Authentication Successful",
    "successUrl":"/openam/console",
    "realm":"/"
}
```

Required: No.

Returning Callback Information to AM

The `/json/authenticate` endpoint supports callback mechanisms to perform complex authentication journeys. Whenever AM needs to return or request information, it will return a JSON object with the authentication step, the authentication identifier, and the related callbacks.

The following types of callbacks are available:

- *Read-only callbacks*. AM uses read-only callbacks to provide information to the user, such as text messages or the amount of time that the user needs to wait before continuing their authentication journey.

- *Interactive callbacks*. AM uses interactive callbacks ask the user for information. For example, to request their user name and password, or to request that the user chooses between different options.
- *Backchannel callbacks*. AM uses backchannel callbacks when it needs to access additional information from the user's request. For example, when it requires a particular header or a certificate.

Read-only and interactive callbacks have an array of `output` elements suitable for displaying to the end user. The JSON returned in interactive callbacks also contains an array of input elements, which must be completed and returned to AM. For example:

```
"output": [
    {
        "name": "prompt",
        "value": " User Name: "
    }
],
"input": [
    {
        "name": "IDToken1",
        "value": ""
    }
]
```

The value of some interactive callbacks can be returned as headers, such as the `X-OpenAM-Username` and `X-OpenAM-Password` headers, but most of them must be returned in JSON as a response to the request.

Depending on how complex the authentication journey is, AM may return several callbacks sequentially. Each must be completed and returned to AM until authentication is successful.

The following example shows a request for authentication, and AM's response of the `NameCallback` and `PasswordCallback` callbacks:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
```

```
{
  "authId": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJvdGsiOiJ...", ❶
  "template": "", ❷
  "stage": "DataStore1", ❸
  "callbacks": [
    {
      "type": "NameCallback", ❹
      "output": [ ❺
        {
          "name": "prompt",
          "value": " User Name: "
        }
      ],
      "input": [ ❻
        {
          "name": "IDToken1",
          "value": ""
        }
      ]
    },
    {
      "type": "PasswordCallback", ❹
      "output": [ ❺
        {
          "name": "prompt",
          "value": " Password: "
        }
      ],
      "input": [ ❻
        {
          "name": "IDToken2",
          "value": ""
        }
      ]
    }
  ]
}
```

Key:

❶ The JWT that uniquely identifies the authentication context to AM.
❷ A template to customize the look of the authentication module, if exists. For more information, see How do I customize the Login page? in the *ForgeRock Knowledge Base*.
❸ The authentication module stage where the authentication journey is at the moment.
❹ The type of callback. It must be one the "Supported Callbacks".
❺ The information AM offers about this callback. Usually, this information would be displayed to the user in the UI.
❻ The information AM is requesting. The user must fill the `"value": ""` object with the required information.

To respond to a callback, send back the whole JSON object with the missing values filled. The following example shows how to respond to the`NameCallback` and `PasswordCallback` callbacks, with the `demo` and `changeit` values filled:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--data '{
   "authId":""eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJvdGsiOiJ...",
   "template":"",
   "stage":"DataStore1",
   "callbacks":[
      {
         "type":"NameCallback",
         "output":[
            {
               "name":"prompt",
               "value":" User Name: "
            }
         ],
         "input":[
            {
               "name":"IDToken1",
               "value":"demo"
            }
         ]
      },
      {
         "type":"PasswordCallback",
         "output":[
            {
               "name":"prompt",
               "value":" Password: "
            }
         ],
         "input":[
            {
               "name":"IDToken2",
               "value":"changeit"
            }
         ]
      }
   ]
}' \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
{
    "tokenId":"AQIC5wM2...U3MTE4NA..*",
    "successUrl": "/openam/console",
    "realm":"/"
}
```

On complex authentication journeys, AM may send several callbacks sequentially. Each must be completed and returned to AM until authentication is successful.

Supported Callbacks

The following types of callbacks are available:

- Interactive Callbacks
- Read-only Callbacks
- Backchannel Callbacks

Interactive Callbacks

AM returns the following callbacks to request information from the user:

ChoiceCallback

Used to display a list of choices and retrieve the selected choice. To indicate that the user selected the first choice, return a value of `0` to AM. For the second choice, return a value of `1`, and so forth. For example:

```
"callbacks":[
   {
      "type":"ChoiceCallback",
      "output":[
         {
            "name":"prompt",
            "value":"Choose one"
         },
         {
            "name":"choices",
            "value":[
               "Choice A",
               "Choice B",
               "Choice C"
            ]
         },
         {
            "name":"defaultChoice",
            "value":2
         }
      ],
      "input":[
         {
            "name":"IDToken1",
            "value":0
         }
      ]
   }
]
```

Class to import: `javax.security.auth.callback.ChoiceCallback`

ConfirmationCallback

Used to ask for a boolean-style confirmation, such as yes/no or true/false, and retrieve the response. Also can present a "Cancel" option. To indicate that the user selected the first choice, return a value of `0` to AM. For the second choice, return a value of `1`, and so forth. For example:

```
"callbacks":[
   {
      "type":"ConfirmationCallback",
      "output":[
         {
            "name":"prompt",
            "value":""
         },
         {
            "name":"messageType",
            "value":0
         },
         {
            "name":"options",
            "value":[
               "Submit",
               "Start Over",
               "Cancel"
            ]
         },
         {
            "name":"optionType",
            "value":-1
         },
         {
            "name":"defaultOption",
            "value":1
         }
      ],
      "input":[
         {
            "name":"IDToken2",
            "value":0
         }
      ]
   }
]
```

Class to import: `javax.security.auth.callback.ConfirmationCallback`

NameCallback

Used to retrieve a data string which can be entered by the user. Usually used for collecting user names. For example:

```
"callbacks":[
   {
      "type":"NameCallback",
      "output":[
         {
            "name":"prompt",
            "value":"User Name"
         }
      ],
      "input":[
         {
            "name":"IDToken1",
            "value":""
         }
      ]
   }
]
```

Class to import: `javax.security.auth.callback.NameCallback`

PasswordCallback

Used to retrieve a password value. For example:

```
 "callbacks":[
   {
      "type":"PasswordCallback",
      "output":[
         {
            "name":"prompt",
            "value":"Password"
         }
      ],
      "input":[
         {
            "name":"IDToken1",
            "value":""
         }
      ]
   }
]
```

Class to import: `javax.security.auth.callback.PasswordCallback`

TextInputCallback

Used to retrieve text input from the end user. For example

```
"callbacks":[
   {
      "type":"TextInputCallback",
      "output":[
         {
            "name":"prompt",
            "value":"User Name"
         }
      ],
      "input":[
         {
            "name":"IDToken1",
            "value":""
         }
      ]
   }
]
```

Class to import: `javax.security.auth.callback.TextInputCallback`

Read-only Callbacks

HiddenValueCallback

Used to return form values that are not visually rendered to the end user. For example:

```
"callbacks":[
   {
      "type":"HiddenValueCallback",
      "output":[
         {
            "name":"value",
            "value":"6186c911-b3be-4dbc-8192-bdf251392072"
         },
         {
            "name":"id",
            "value":"jwt"
         }
      ],
      "input":[
         {
            "name":"IDToken1",
            "value":"jwt"
         }
      ]
   }
]
```

Class to import: `com.sun.identity.authentication.callbacks.HiddenValueCallback`

MetadataCallback

Used to inject key-value meta data into the authentication process. For example:

```

"callbacks":[
   {
      "type":"MetadataCallback",
      "output":[
         {
            "name":"data",
            "value":{
                  "myParameter": "MyValue"
            }
         }
      ]
   }
]
```

Class to import: `com.sun.identity.authentication.spi.MetadataCallback`

PollingWaitCallback

Tells the user the amount of time to wait before responding to the callback.

```
"callbacks":[
   {
      "type":"PollingWaitCallback",
      "output":[
         {
            "name":"waitTime",
            "value":"8000"
         },
         {
            "name":"message",
            "value":"Waiting for response..."
         }
      ]
   }
]
```

Class to import: `org.forgerock.openam.authentication.callbacks.PollingWaitCallback`

RedirectCallback

Used to redirect the user's browser or user-agent.

```
"callbacks":[
   {
      "type":"RedirectCallback",
      "output":[
         {
            "name":"redirectUrl",
            "value":"https://accounts.google.com/o/oauth2/v2/auth?nonce..."
         },
         {
            "name":"redirectMethod",
            "value":"GET"
         },
         {
            "name":"trackingCookie",
            "value":true
         }
      ]
   }
]
```

Class to import: `com.sun.identity.authentication.spi.RedirectCallback`

TextOutputCallback

Used to display a message to the end user.

```
"callbacks":[
   {
      "type":"TextOutputCallback",
      "output":[
         {
            "name":"message",
            "value":"Default message"
         },
         {
            "name":"messageType",
            "value":"0"
         }
      ]
   }
]
```

Class to import: `javax.security.auth.callback.TextOutputCallback`

Backchannel Callbacks

AM uses backchannel callbacks when it needs to recover additional information from the user's request. For example, when it requires a particular header or a certificate.

HttpCallback

Used to access user credentials sent in the Authorization header. For example:

```
Authorization: Basic bXlDbGllbnQ6Zm9yZ2Vyb2Nr
```

Class to import: `com.sun.identity.authentication.spi.HttpCallback`

LanguageCallback

Used to retrieve the locale for localizing text presented to the end user. The locale is sent in the request as a header.

Class to import: `javax.security.auth.callback.LanguageCallback`

ScriptTextOutputCallback

Used to insert a script into the page presented to the end user. The script can, for example, collect data about the user's environment.

Class to import: `com.sun.identity.authentication.callbacks.ScriptTextOutputCallback`

X509CertificateCallback

Used to retrieve the content of an x.509 certificate, for example, from a header.

Class to import: `com.sun.identity.authentication.spi.X509CertificateCallback`

Logging out of AM Using REST

Authenticated users can log out with the token cookie value and an HTTP POST to `/json/sessions/?_action=logout`:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Cache-Control: no-cache" \
--header "iplanetDirectoryPro: AQIC5wM2...U3MTE4NA..*" \
--header "Accept-API-Version: resource=3.1, protocol=1.0" \
https://openam.example.com:8443/openam/json/realms/root/sessions/?_action=logout
{
    "result":"Successfully logged out"
}
```

Invalidating All Sessions for a Given User

To log out all sessions for a given user, first obtain a list of session handles of their active sessions, by performing an HTTP GET to the `/json/sessions/` endpoint, using the SSO token of an administrative user, such as `amAdmin` as the value of the `iPlanetDirectoryPro` header. You must also specify a `queryFilter` parameter.

The `queryFilter` parameter requires the name of the user, and the realm to search. For example, to obtain a list of session handles for a user named `demo` in the top-level realm, the query filter value would be:

```
username eq "demo" and realm eq "/"
```

Note

The query filter value must be URL encoded when sent over HTTP.

For more information on query filter parameters, see "Query".

In the following example, there are two active sessions:

```
$ curl \
--request GET \
--header "Content-Type: application/json" \
--header "Cache-Control: no-cache" \
--header "iPlanetDirectoryPro: AQICS...NzEz*" \
--header "Accept-API-Version: resource=3.1, protocol=1.0" \
https://openam.example.com:8443/openam/json/realms/root/sessions?_queryFilter=username%20eq%20%22demo
%22%20and%20realm%20eq%20%22%2F%22
{
    "result": [
        {
            "username": "demo",
            "universalId": "id=demo,ou=user,dc=openam,dc=forgerock,dc=org",
            "realm":"/",
            "sessionHandle":"shandle:SJ80.*AA....JT.*",
            "latestAccessTime":"2018-10-23T09:37:54.387Z",
            "maxIdleExpirationTime":"2018-10-23T10:07:54Z",
            "maxSessionExpirationTime":"2018-10-23T11:37:54Z"
        },
         {
            "username": "demo",
            "universalId": "id=demo,ou=user,dc=openam,dc=forgerock,dc=org",
            "realm":"/",
            "sessionHandle":"shandle:H4CV.*DV....FM.*",
            "latestAccessTime":"2018-10-23T09:37:43.780Z",
            "maxIdleExpirationTime":"2018-10-23T10:07:43Z",
            "maxSessionExpirationTime":"2018-10-23T11:37:43Z"
        }
    ],
    "resultCount": 2,
    "pagedResultsCookie": null,
    "totalPagedResultsPolicy": "NONE",
    "totalPagedResults": -1,
    "remainingPagedResults": -1
}
```

To log out all sessions for the specific user, perform an HTTP POST to the `/json/sessions/` endpoint, using the SSO token of an administrative user, such as `amAdmin` as the value of the `iPlanetDirectoryPro` header. You must also specify the `logoutByHandle` action, and include an array of the session handles to invalidate in the POST body, in a property named `sessionHandles`, as shown below:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Cache-Control: no-cache" \
--header "iplanetDirectoryPro: AQIC5w...NTcy*" \
--header "Accept-API-Version: resource=3.1, protocol=1.0" \
--data '{
    "sessionHandles": [
        "shandle:SJ80.*AA....JT.*",
        "shandle:H4CV.*DV....FM.*"
    ]
}' \
https://openam.example.com:8443/openam/json/realms/root/sessions/?_action=logoutByHandle
{
    "result": {
        "shandle:SJ80.*AA....JT.*": true,
        "shandle:H4CV.*DV....FM.*": true
    }
}
```

Load Balancer and Proxy Layer Requirements

When authentication depends on the client IP address and AM lies behind a load balancer or proxy layer, configure the load balancer or proxy to send the address by using the `X-Forwarded-For` header, and configure AM to consume and forward the header as necessary. For details, see "Handling HTTP Request Headers" in the *Installation Guide*.

Windows Desktop SSO Requirements

When authenticating with Windows Desktop SSO, add an `Authorization` header containing the string `Basic `, followed by a base64-encoded string of the username, a colon character, and the password. In the following example, the credentials `demo:changeit` are base64-encoded into the string `ZGVtbzpjaGFuZ2VpdA==`:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--header "X-OpenAM-Username: demo" \
--header "X-OpenAM-Password: changeit" \
--header "Authorization: Basic ZGVtbzpjaGFuZ2VpdA==" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
{
    "tokenId":"AQIC5w...NTcy*",
    "successUrl":"/openam/console",
    "realm":"/"
}
```

Using the Session Token After Authentication

The following is a common scenario when accessing AM by using REST API calls:

- First, call the `/json/authenticate` endpoint to log a user in to AM. This REST API call returns a `tokenID` value, which is used in subsequent REST API calls to identify the user:

  ```
  $ curl \
  --request POST \
  --header "Content-Type: application/json" \
  --header "X-OpenAM-Username: demo" \
  --header "X-OpenAM-Password: changeit" \
  --header "Accept-API-Version: resource=2.0, protocol=1.0" \
  'https://openam.example.com:8443/openam/json/realms/root/authenticate'
  {
      "tokenId":"AQIC5wM...TU3OQ*",
      "successUrl":"/openam/console",
      "realm":"/"
  }
  ```

  The returned `tokenID` is known as a session token (also referred to as an SSO token). REST API calls made after successful authentication to AM must present the session token in the HTTP header as proof of authentication.

- Next, call one or more additional REST APIs on behalf of the logged-in user. Each REST API call passes the user's `tokenID` back to AM in the HTTP header as proof of previous authentication.

  The following is a *partial* example of a **curl** command that inserts the token ID returned from a prior successful AM authentication attempt into the HTTP header:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5w...NTcy*" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--data '{
  ...
```

Observe that the session token is inserted into a header field named `iPlanetDirectoryPro`. This header field name must correspond to the name of the AM session cookie—by default, `iPlanetDirectoryPro`. You can find the cookie name in the AM console by navigating to Deployment > Servers > *Server Name* > Security > Cookie, in the Cookie Name field of the AM console.

Once a user has authenticated, it is *not* necessary to insert login credentials in the HTTP header in subsequent REST API calls. Note the absence of `X-OpenAM-Username` and `X-OpenAM-Password` headers in the preceding example.

Users are required to have appropriate privileges in order to access AM functionality using the REST API. For example, users who lack administrative privileges cannot create AM realms. For more information on the AM privilege model, see "Delegating Realm Administration Privileges" in the *Setup and Maintenance Guide*.

- Finally, call the REST API to log the user out of AM as described in "Authentication and Logout using REST". As with other REST API calls made after a user has authenticated, the REST API call to log out of AM requires the user's `tokenID` in the HTTP header.

Server Information

You can retrieve AM server information by using HTTP GET on `/json/serverinfo/*` as follows:

```
$ curl \
--request GET \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.1, protocol=1.0" \
https://openam.example.com:8443/openam/json/serverinfo/*
{
    "domains": [
        ".example.com"
    ],
    "protectedUserAttributes": [],
    "cookieName": "iPlanetDirectoryPro",
    "secureCookie": false,
    "forgotPassword": "false",
    "forgotUsername": "false",
    "kbaEnabled": "false",
    "selfRegistration": "false",
    "lang": "en-US",
    "successfulUserRegistrationDestination": "default",
    "socialImplementations": [
        {
            "iconPath": "XUI/images/logos/facebook.png",
            "authnChain": "FacebookSocialAuthenticationService",
            "displayName": "Facebook",
            "valid": true
        }
    ],
    "referralsEnabled": "false",
    "zeroPageLogin": {
        "enabled": false,
        "refererWhitelist": [
            ""
        ],
        "allowedWithoutReferer": true
    },
    "realm": "/",
    "xuiUserSessionValidationEnabled": true,
    "FQDN": "openam.example.com"
}
```

Token Encoding

Valid tokens in AM require configuration either in percent encoding or in *C66Encode* format. C66Encode format is encouraged. It is the default token format for AM, and is used in this section. The following is an example token that has not been encoded:

```
AQIC5wM2LY4SfczntBbXvEAOuECbqMY3J4NW3byH6xwgkGE=@AAJTSQACMDE=#
```

This token includes reserved characters such as `+`, `/`, and `=` (The `@`, `#`, and `*` are not reserved characters per se, but substitutions are still required). To c66encode this token, you would substitute certain characters for others, as follows:

+ is replaced with -
/ is replaced with _
= is replaced with .
@ is replaced with *
# is replaced with *
* (first instance) is replaced with @
* (subsequent instances) is replaced with #

In this case, the translated token would appear as shown here:

```
AQIC5wM2LY4SfczntBbXvEAOuECbqMY3J4NW3byH6xwgkGE.*AAJTSQACMDE.*
```

Logging

AM supports two Audit Logging Services: a new common REST-based Audit Logging Service, and the legacy Logging Service, which is based on a Java SDK and is available in AM versions prior to OpenAM 13. The legacy Logging Service is deprecated.

Both audit facilities log AM REST API calls.

Common Audit Logging of REST API Calls

AM logs information about all REST API calls to the `access` topic. For more information about AM audit topics, see "Audit Log Topics" in the *Setup and Maintenance Guide*.

Locate specific REST endpoints in the `http.path` log file property.

Legacy Logging of REST API Calls

AM logs information about REST API calls to two files:

- **amRest.access**. Records accesses to a CREST endpoint, regardless of whether the request successfully reached the endpoint through policy authorization.

  An `amRest.access` example is as follows:

  ```
  $ cat openam/openam/log/amRest.access
  #Version: 1.0
  #Fields: time  Data  LoginID  ContextID  IPAddr  LogLevel  Domain  LoggedBy  MessageID  ModuleName
  NameID  HostName
  "2011-09-14 16:38:17"   /home/user/openam/openam/log/ "cn=dsameuser,ou=DSAME Users,o=openam"
  aa307b2dcb721d4201 "Not Available" INFO  o=openam   "cn=dsameuser,ou=DSAME Users,o=openam"
  LOG-1  amRest.access  "Not Available"  192.168.56.2
  "2011-09-14 16:38:17"  "Hello World"  id=bjensen,ou=user,o=openam 8a4025a2b3af291d01  "Not Available"
  INFO  o=openam id=amadmin,ou=user,o=openam "Not Available" amRest.access "Not Available"
  192.168.56.2
  ```

- **amRest.authz**. Records all CREST authorization results regardless of success. If a request has an entry in the `amRest.access` log, but no corresponding entry in `amRest.authz`, then that endpoint was not protected by an authorization filter and therefore the request was granted access to the resource.

  The `amRest.authz` file contains the `Data` field, which specifies the authorization decision, resource, and type of action performed on that resource. The `Data` field has the following syntax:

```
("GRANT"||"DENY") > "RESOURCE | ACTION"

where
  "GRANT > " is prepended to the entry if the request was allowed
  "DENY  > " is prepended to the entry if the request was not allowed
  "RESOURCE" is "ResourceLocation | ResourceParameter"
     where
       "ResourceLocation" is the endpoint location (e.g., subrealm/applicationtypes)
       "ResourceParameter" is the ID of the resource being touched
        (e.g., myApplicationType) if applicable. Otherwise, this field is empty
        if touching the resource itself, such as in a query.

  "ACTION" is "ActionType | ActionParameter"
     where
       "ActionType" is "CREATE||READ||UPDATE||DELETE||PATCH||ACTION||QUERY"
       "ActionParameter" is one of the following depending on the ActionType:
          For CREATE: the new resource ID
          For READ: empty
          For UPDATE: the revision of the resource to update
          For DELETE: the revision of the resource to delete
          For PATCH: the revision of the resource to patch
          For ACTION: the actual action performed (e.g., "forgotPassword")
          For QUERY: the query ID if any
```

```
$ cat openam/openam/log/amRest.authz
#Version: 1.0
#Fields: time   Data  ContextID  LoginID  IPAddr  LogLevel  Domain  MessageID  LoggedBy  NameID
ModuleName    HostName
"2014-09-16 14:17:28"   /var/root/openam/openam/log/   7d3af9e799b6393301
"cn=dsameuser,ou=DSAME Users,dc=openam,dc=forgerock,dc=org" "Not Available" INFO
dc=openam,dc=forgerock,dc=org  LOG-1  "cn=dsameuser,ou=DSAME Users,dc=openam,dc=forgerock,dc=org"
"Not Available" amRest.authz    10.0.1.5
"2014-09-16 15:56:12"  "GRANT > sessions|ACTION|logout|AdminOnlyFilter"  d3977a55a2ee18c201
id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org "Not Available" INFO  dc=openam,dc=forgerock,dc=org
OAuth2Provider-2  "cn=dsameuser,ou=DSAME Users,dc=openam,dc=forgerock,dc=org"  "Not Available"
amRest.authz    127.0.0.1
"2014-09-16 15:56:40"   "GRANT > sessions|ACTION|logout|AdminOnlyFilter"  eedbc205bf51780001
id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org  "Not Available" INFO dc=openam,dc=forgerock,dc=org
OAuth2Provider-2  "cn=dsameuser,ou=DSAME Users,dc=openam,dc=forgerock,dc=org"  "Not Available"
amRest.authz    127.0.0.1
```

AM also provides additional information in its debug notifications for accesses to any endpoint, depending on the message type (error, warning or message) including realm, user, and result of the operation.

Reference

This reference section covers return codes and system settings relating to REST API support in AM.

REST APIs

amster service name: `RestApis`

The following settings are available in this service:

Default Resource Version

The API resource version to use when the REST request does not specify an explicit version. Choose from:

- `Latest`. If an explicit version is not specified, the latest resource version of an API is used.
- `Oldest`. If an explicit version is not specified, the oldest supported resource version of an API is used. Note that since APIs may be deprecated and fall out of support, the oldest *supported* version may not be the first version.
- `None`. If an explicit version is not specified, the request will not be handled and an error status is returned.

The possible values for this property are:

- `Latest`
- `Oldest`
- `None`

Default value: `Latest`

amster attribute: `defaultVersion`

Warning Header

Whether to include a warning header in the response to a request which fails to include the `Accept-API-Version` header.

Default value: `false`

amster attribute: `warningHeader`

API Descriptions

Whether API Explorer and API Docs are enabled in AM and how the documentation for them is generated. Dynamic generation includes descriptions from any custom services and authentication modules you may have added. Static generation only includes services and authentication modules that were present when AM was built. Note that dynamic documentation generation may not work in some application containers.

The possible values for this property are:

- `DYNAMIC`. Enabled with Dynamic Documentation
- `STATIC`. Enabled with Static Documentation

- `DISABLED`

Default value: `STATIC`

amster attribute: `descriptionsState`

Default Protocol Version

The API protocol version to use when a REST request does not specify an explicit version. Choose from:

- `Oldest`. If an explicit version is not specified, the oldest protocol version is used.
- `Latest`. If an explicit version is not specified, the latest protocol version is used.
- `None`. If an explicit version is not specified, the request will not be handled and an error status is returned.

The possible values for this property are:

- `Oldest`
- `Latest`
- `None`

Default value: `Latest`

amster attribute: `defaultProtocolVersion`

Enable CSRF Protection

If enabled, all non-read/query requests will require the X-Requested-With header to be present.

Requiring a non-standard header ensures requests can only be made via methods (XHR) that have stricter same-origin policy protections in Web browsers, preventing Cross-Site Request Forgery (CSRF) attacks. Without this filter, cross-origin requests are prevented by the use of the application/json Content-Type header, which is less robust.

Default value: `true`

amster attribute: `csrfFilterEnabled`

Appendix B. About Scripting

You can use scripts for client-side and server-side authentication, policy conditions, and handling OpenID Connect claims.

The Scripting Environment

AM supports scripts written in either JavaScript, or Groovy [1], and the same variables and bindings are delivered to scripts of either language.

+ *How to determine the JavaScript Engine Version?*

You can use a script to check the version of the JavaScript engine AM is using. You could temporarily add the following script to a Scripted Decision node, for example, to output the engine version to the debug log:

```
var rhino = JavaImporter(
  org.mozilla.javascript.Context
)

var currentContext = rhino.Context.getCurrentContext()
var rhinoVersion = currentContext.getImplementationVersion()

logger.error("JS Script Engine: " + rhinoVersion)

outcome = "true"
```

[1] Scripts used for client-side authentication must be in written in JavaScript.

Note

Ensure the following are listed in the Java class whitelist property of the scripting engine.

- `org.mozilla.javascript.Context`
- `org.forgerock.openam.scripting.timeouts.*`

To view the Java class whitelist, go to Configure > Global Services > Scripting > Secondary Configurations. Select the script type, and on the Secondary Configurations tab, click engineConfiguration.

For information on the capabilities of the JavaScript engine AM uses, see Mozilla Rhino.

+ *How to determine the Groovy Engine Version?*

You can use a script to check the version of the Groovy scripting engine AM is using. You could temporarily add the following script to a Scripted Decision node, for example, to output the engine version to the debug log:

```
logger.error("Groovy Script Engine: " + GroovySystem.version)

outcome = "true"
```

Note

Ensure the following are listed in the Java class whitelist property of the scripting engine.

- `groovy.lang.GroovySystem`

To view the Java class whitelist, go to Configure > Global Services > Scripting > Secondary Configurations. Select the script type, and on the Secondary Configurations tab, click engineConfiguration.

For information on the capabilities of the Groovy engine AM uses, see Apache Groovy.

To access the functionality AM provides, import the required Java class or package, as follows:

JavaScript

```
var fr = JavaImporter(
    org.forgerock.openam.auth.node.api,
    javax.security.auth.callback.NameCallback
);
with (fr) {
    ...
}
```

Groovy

```
import org.forgerock.openam.auth.node.api.*;
import javax.security.auth.callback.NameCallback;
```

You may need to whitelist the classes you use in scripts. See "Security".

You can use scripts to modify default AM behavior in the following situations, also known as *contexts*:

Client-side Authentication

Scripts that are executed on the client during authentication. Client-side scripts must be in JavaScript.

Server-side Authentication

Scripts are included in an authentication module within a chain and are executed on the server during authentication.

Authentication Trees

Scripts are included in an authentication node within a tree and are executed on the server during authentication.

Policy Condition

Scripts used as conditions within policies.

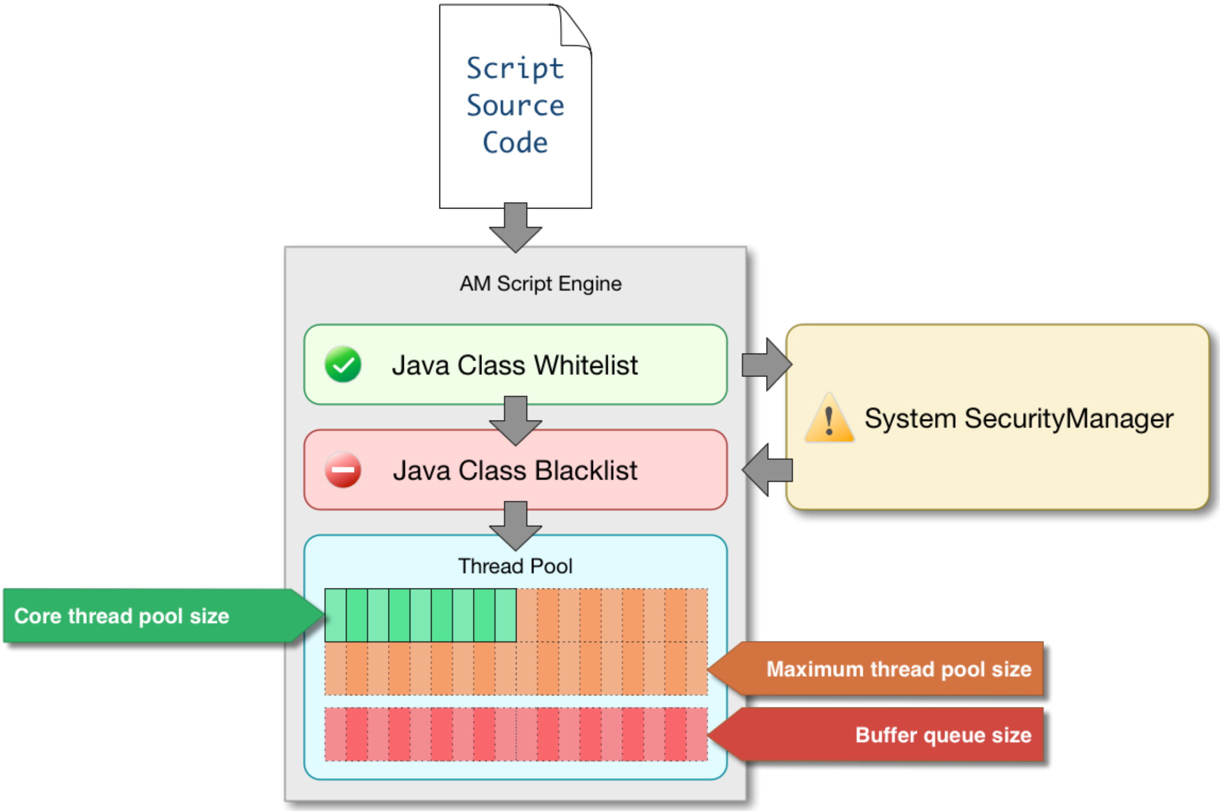
OIDC Claims

Scripts that gather and populate the claims in a request when issuing an ID token or making a request to the `userinfo` endpoint.

For information on the global API, available to all script types, see "Global Scripting API Functionality".

AM implements a configurable scripting engine for each of the context types that are executed on the server.

The scripting engines in AM have two main components: security settings, and the thread pool.



Security

AM scripting engines provide security features for ensuring that malicious Java classes are not directly called. The engines validate scripts by checking all directly-called Java classes against a configurable blacklist and whitelist, and, optionally, against the JVM SecurityManager, if it is configured.

Whitelists and blacklists contain class names that are allowed or denied execution respectively. Specify classes in whitelists and blacklists by name or by using regular expressions.

Classes called by the script are checked against the whitelist first, and must match at least one pattern in the list. The blacklist is applied after the whitelist, and classes matching any pattern are disallowed.

You can also configure the scripting engine to make an additional call to the JVM security manager for each class that is accessed. The security manager throws an exception if a class being called is not allowed to execute.

For more information on configuring script engine security, see "Scripting".

Important Points About Script Engine Security

The following points should be considered when configuring the security settings within each script engine:

The scripting engine only validates directly accessible classes.

The security settings only apply to classes that the script *directly* accesses. If the script calls `Foo.a()` and then that method calls `Bar.b()`, the scripting engine will be unable to prevent it. You must consider the whole chain of accessible classes.

> **Note**
>
> *Access* includes actions such as:
>
> - Importing or loading a class.
> - Accessing any instance of that class. For example, passed as a parameter to the script.
> - Calling a static method on that class.
> - Calling a method on an instance of that class.
> - Accessing a method or field that returns an instance of that class.

Potentially dangerous Java classes are blacklisted by default.

All Java reflection classes (`java.lang.Class`, `java.lang.reflect.*`) are blacklisted by default to avoid bypassing the security settings.

The `java.security.AccessController` class is also blacklisted by default to prevent access to the `doPrivileged()` methods.

> **Caution**
>
> You should not remove potentially dangerous Java classes from the blacklist.

The whitelists and blacklists match class or package names only.

The whitelist and blacklist patterns apply only to the exact class or package names involved. The script engine does not know anything about inheritance, so it is best to whitelist known, specific classes.

Thread Pools

Each script is executed in an individual thread. Each scripting engine starts with an initial number of threads available for executing scripts. If no threads are available for execution, AM creates a new thread to execute the script, until the configured maximum number of threads is reached.

If the maximum number of threads is reached, pending script executions are queued in a number of buffer threads, until a thread becomes available for execution. If a created thread has completed script execution and has remained idle for a configured amount of time, AM terminates the thread, shrinking the pool.

For more information on configuring script engine thread pools, see "Scripting".

Global Scripting API Functionality

This section covers functionality available to each of the server-side script types.

Global API functionality includes:

- Accessing HTTP Services
- Debug Logging

Accessing HTTP Services

AM passes an HTTP client object, `httpClient`, to server-side scripts. Server-side scripts can call HTTP services with the `httpClient.send` method. The method returns an `HttpClientResponse` object.

Configure the parameters for the HTTP client object by using the `org.forgerock.http.protocol` package. This package contains the `Request` class, which has methods for setting the URI and type of request.

The following example, taken from the default server-side Scripted authentication module script, uses these methods to call an online API to determine the longitude and latitude of a user based on their postal address:

```
function getLongitudeLatitudeFromUserPostalAddress() {

    var request = new org.forgerock.http.protocol.Request();

    request.setUri("http://maps.googleapis.com/maps/api/geocode/json?address=" +
 encodeURIComponent(userPostalAddress));
    request.setMethod("GET");

    var response = httpClient.send(request).get();
    logResponse(response);

    var geocode = JSON.parse(response.getEntity());
    var i;

    for (i = 0; i < geocode.results.length; i++) {
        var result = geocode.results[i];
        latitude = result.geometry.location.lat;
        longitude = result.geometry.location.lng;

        logger.message("latitude:" + latitude + " longitude:" + longitude);
    }
}
```

HTTP client requests are synchronous and blocking until they return. You can, however, set a global timeout for server-side scripts. For details, see "Scripted Authentication Module Properties".

Server-side scripts can access response data by using the methods listed in the table below.

HTTP Client Response Methods

Method	Parameters	Return Type	Description
`HttpClientResponse.getCookies`	`Void`	`Map<String, String>`	Get the cookies for the returned response, if any exist.
`HttpClientResponse.getEntity`	`Void`	`String`	Get the entity of the returned response.
`HttpClientResponse.getHeaders`	`Void`	`Map<String, String>`	Get the headers for the returned response, if any exist.
`HttpClientResponse.getReasonPhrase`	`Void`	`String`	Get the reason phrase of the returned response.
`HttpClientResponse.getStatusCode`	`Void`	`Integer`	Get the status code of the returned response.
`HttpClientResponse.hasCookies`	`Void`	`Boolean`	Indicate whether the returned response had any cookies.
`HttpClientResponse.hasHeaders`	`Void`	`Boolean`	Indicate whether the returned response had any headers.

Debug Logging

Server-side scripts can write messages to AM debug logs by using the `logger` object.

AM does not log debug messages from scripts by default. You can configure AM to log such messages by setting the debug log level for the `amScript` service. For details, see "Debug Logging By Service" in the *Setup and Maintenance Guide*.

The following table lists the `logger` methods.

Logger Methods

Method	Parameters	Return Type	Description
`logger.error`	*`Error Message`* (type: `String`)	`Void`	Write *Error Message* to AM debug logs if ERROR level logging is enabled.

Method	Parameters	Return Type	Description
`logger.errorEnabled`	`Void`	`Boolean`	Return `true` when ERROR level debug messages are enabled.
`logger.message`	*`Message`* (type: `String`)	`Void`	Write *Message* to AM debug logs if MESSAGE level logging is enabled.
`logger.messageEnabled`	`Void`	`Boolean`	Return `true` when MESSAGE level debug messages are enabled.
`logger.warning`	*`Warning Message`* (type: `String`)	`Void`	Write *Warning Message* to AM debug logs if WARNING level logging is enabled.
`logger.warningEnabled`	`Void`	`Boolean`	Return `true` when WARNING level debug messages are enabled.

Managing Scripts

This section shows you how to manage scripts used for client-side and server-side scripted authentication, custom policy conditions, and handling OpenID Connect claims using the AM console, the **ssoadm** command, and the REST API.

Managing Scripts With the AM Console

The following procedures describe how to create, modify, and delete scripts using the AM console:

- "To Create Scripts by Using the AM Console"
- "To Modify Scripts by Using the AM Console"
- "To Delete Scripts by Using the AM Console"

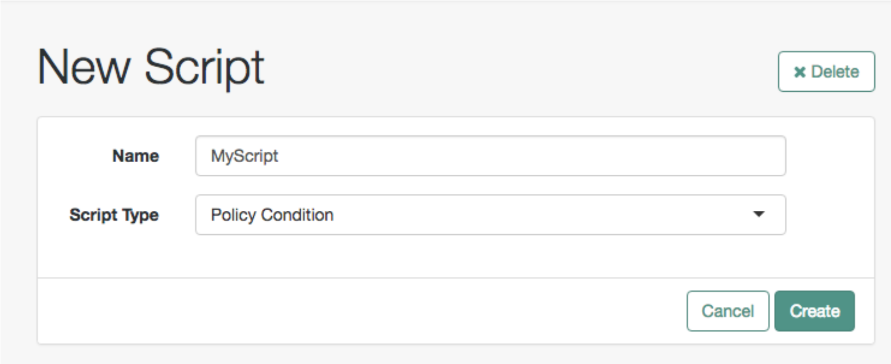
To Create Scripts by Using the AM Console

1. Log in to the AM console as an AM administrator, for example, `amadmin`.

2. Navigate to Realms > *Realm Name* > Scripts.

3. Click New Script.

   The New Script page appears:



New Script

Name: MyScript

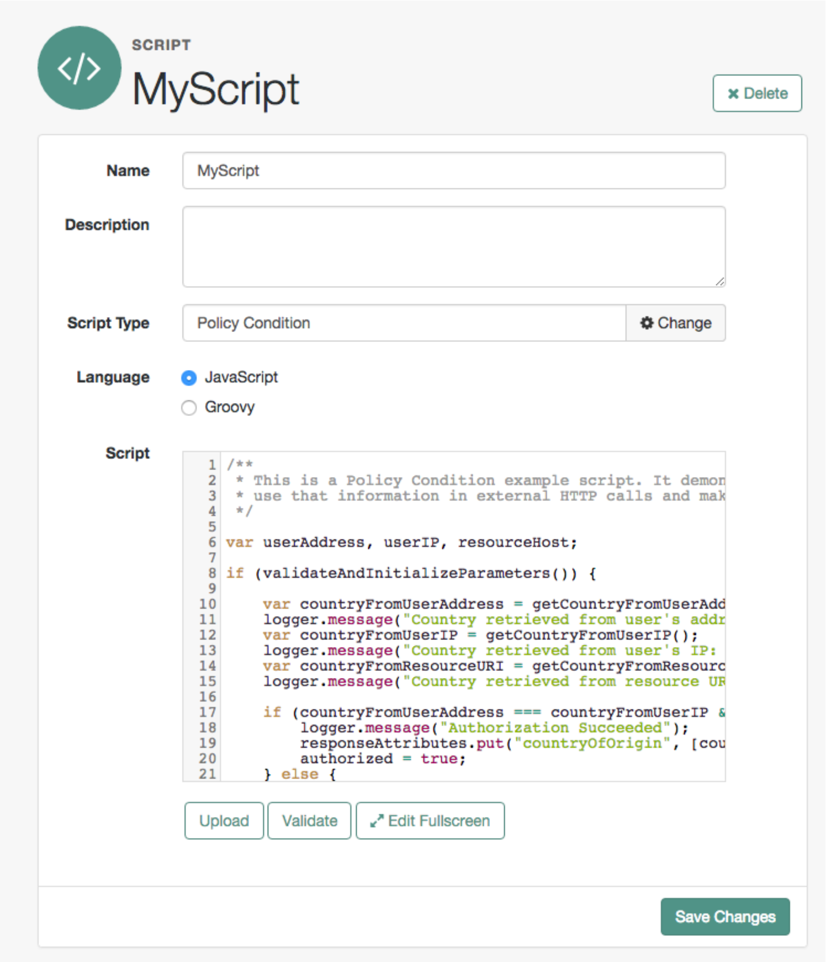
Script Type: Policy Condition

4. Specify a name for the script.

5. Select the type of script from the Script Type drop-down list.

6. Click Create.

   The *Script Name* page appears:



Name: MyScript

Description:

Script Type: Policy Condition ⚙ Change

Language: ● JavaScript ○ Groovy

Script

```
/**
 * This is a Policy Condition example script. It demon
 * use that information in external HTTP calls and mak
 */

var userAddress, userIP, resourceHost;

if (validateAndInitializeParameters()) {

    var countryFromUserAddress = getCountryFromUserAdd
    logger.message("Country retrieved from user's addr
    var countryFromUserIP = getCountryFromUserIP();
    logger.message("Country retrieved from user's IP:
    var countryFromResourceURI = getCountryFromResourc
    logger.message("Country retrieved from resource UR

    if (countryFromUserAddress === countryFromUserIP &
        logger.message("Authorization Succeeded");
        responseAttributes.put("countryOfOrigin", [cou
        authorized = true;
    } else {
```

7. Enter values on the *Script Name* page as follows:

   a. Enter a description of the script.

   b. Choose the script language, either JavaScript or Groovy. Note that not every script type supports both languages.

   c. Enter the source code in the Script field.

      On supported browsers, you can click Upload, navigate to the script file, and then click Open to upload the contents to the Script field.

   d. Click Validate to check for compilation errors in the script.

Correct any compilation errors, and revalidate the script until all errors have been fixed.

e. Save your changes.

To Modify Scripts by Using the AM Console

1. Log in to the AM console as an AM administrator, for example, `amadmin`.
2. Navigate to Realms > *Realm Name* > Scripts.
3. Select the script you want to modify from the list of scripts.

   The *Script Name* page appears.
4. Modify values on the *Script Name* page as needed. Note that if you change the Script Type, existing code in the script is replaced.
5. If you modified the code in the script, click Validate to check for compilation errors.

   Correct any compilation errors, and revalidate the script until all errors have been fixed.
6. Save your changes.

To Delete Scripts by Using the AM Console

1. Log in to the AM console as an AM administrator, for example, `amadmin`.
2. Navigate to Realms > *Realm Name* > Scripts.
3. Choose one or more scripts to delete by activating the checkboxes in the relevant rows. Note that you can only delete user-created scripts—you cannot delete the global sample scripts provided with AM.
4. Click Delete.

Managing Scripts With the REST API

This section shows you how to manage scripts used for client-side and server-side scripted authentication, custom policy conditions, and handling OpenID Connect claims by using the REST API.

AM provides the `scripts` REST endpoint for the following:

- "Querying Scripts"
- "Reading a Script"

- "Validating a Script"
- "Creating a Script"
- "Updating a Script"
- "Deleting a Script"

User-created scripts are realm-specific, hence the URI for the scripts' API can contain a realm component, such as `/json{/realm}/scripts`. If the realm is not specified in the URI, the top level realm is used.

> **Tip**
>
> AM includes some global example scripts that can be used in any realm.

Scripts are represented in JSON and take the following form. Scripts are built from standard JSON objects and values (strings, numbers, objects, sets, arrays, `true`, `false`, and `null`). Each script has a system-generated *universally unique identifier* (UUID), which must be used when modifying existing scripts. Renaming a script will not affect the UUID:

```
{
  "_id": "7e3d7067-d50f-4674-8c76-a3e13a810c33",
  "name": "Scripted Module - Server Side",
  "description": "Default global script for server side Scripted Authentication Module",
  "script": "dmFyIFNUQVJUX1RJ...",
  "language": "JAVASCRIPT",
  "context": "AUTHENTICATION_SERVER_SIDE",
  "createdBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
  "creationDate": 1433147666269,
  "lastModifiedBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedDate": 1433147666269
}
```

The values for the fields shown in the example above are explained below:

`_id`

The UUID that AM generates for the script.

`name`

The name provided for the script.

`description`

An optional text string to help identify the script.

`script`

The source code of the script. The source code is in UTF-8 format and encoded into Base64.

For example, a script such as the following:

```
var a = 123;
var b = 456;
```

When encoded into Base64 becomes:

```
dmFyIGEgPSAxMjM7IA0KdmFyIGIgPSA0NTY7
```

`language`

The language the script is written in - `JAVASCRIPT` or `GROOVY`.

Language Support per Context

Script Context	Supported Languages
`POLICY_CONDITION`	`JAVASCRIPT`, `GROOVY`
`AUTHENTICATION_SERVER_SIDE`	`JAVASCRIPT`, `GROOVY`
`AUTHENTICATION_CLIENT_SIDE`	`JAVASCRIPT`
`OIDC_CLAIMS`	`JAVASCRIPT`, `GROOVY`
`AUTHENTICATION_TREE_DECISION_NODE`	`JAVASCRIPT`, `GROOVY`

`context`

The context type of the script.

Supported values are:

`POLICY_CONDITION`

Policy Condition

`AUTHENTICATION_SERVER_SIDE`

Server-side Authentication

`AUTHENTICATION_CLIENT_SIDE`

Client-side Authentication

> **Note**
>
> Client-side scripts must be written in JavaScript.

`OIDC_CLAIMS`

OIDC Claims

`AUTHENTICATION_TREE_DECISION_NODE`

Authentication scripts used by Scripted Tree Decision authentication nodes.

`createdBy`

A string containing the universal identifier DN of the subject that created the script.

`creationDate`

An integer containing the creation date and time, in ISO 8601 format.

`lastModifiedBy`

A string containing the universal identifier DN of the subject that most recently updated the resource type.

If the script has not been modified since it was created, this property will have the same value as `createdBy`.

`lastModifiedDate`

A string containing the last modified date and time, in ISO 8601 format.

If the script has not been modified since it was created, this property will have the same value as `creationDate`.

Querying Scripts

To list all the scripts in a realm, as well as any global scripts, perform an HTTP GET to the `/json{/*realm*}/scripts` endpoint with a `_queryFilter` parameter set to `true`.

Note

If the realm is not specified in the URL, AM returns scripts in the top level realm, as well as any global scripts.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.1" \
https://openam.example.com:8443/openam/json/realms/root/realms/myrealm/scripts?_queryFilter=true
{
    "result": [
        {
            "_id": "9de3eb62-f131-4fac-a294-7bd170fd4acb",
            "name": "Scripted Policy Condition",
            "description": "Default global script for Scripted Policy Conditions",
            "script": "LyoqCiAqIFRoaXMg...",
            "language": "JAVASCRIPT",
            "context": "POLICY_CONDITION",
            "createdBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
            "creationDate": 1433147666269,
            "lastModifiedBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
            "lastModifiedDate": 1433147666269
        },
        {
            "_id": "7e3d7067-d50f-4674-8c76-a3e13a810c33",
            "name": "Scripted Module - Server Side",
            "description": "Default global script for server side Scripted Authentication Module",
            "script": "dmFyIFNUQVJUX1RJ...",
            "language": "JAVASCRIPT",
            "context": "AUTHENTICATION_SERVER_SIDE",
            "createdBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
            "creationDate": 1433147666269,
            "lastModifiedBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
            "lastModifiedDate": 1433147666269
        }
    ],
    "resultCount": 2,
    "pagedResultsCookie": null,
    "remainingPagedResults": -1
}
```

Supported _queryFilter Fields and Operators

Field	Supported Operators
`_id`	Equals (`eq`), Contains (`co`), Starts with (`sw`)
`name`	Equals (`eq`), Contains (`co`), Starts with (`sw`)
`description`	Equals (`eq`), Contains (`co`), Starts with (`sw`)
`script`	Equals (`eq`), Contains (`co`), Starts with (`sw`)
`language`	Equals (`eq`), Contains (`co`), Starts with (`sw`)
`context`	Equals (`eq`), Contains (`co`), Starts with (`sw`)

Reading a Script

To read an individual script in a realm, perform an HTTP GET using the `/json{/realm}/scripts` endpoint, specifying the UUID in the URL.

> **Tip**
>
> To read a script in the top-level realm, or to read a built-in global script, do not specify a realm in the URL.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.1" \
https://openam.example.com:8443/openam/json/realms/root/realms/myrealm/scripts/9de3eb62-f131-4fac-
a294-7bd170fd4acb
{
    "_id": "9de3eb62-f131-4fac-a294-7bd170fd4acb",
    "name": "Scripted Policy Condition",
    "description": "Default global script for Scripted Policy Conditions",
    "script": "LyoqCiAqIFRoaXMg...",
    "language": "JAVASCRIPT",
    "context": "POLICY_CONDITION",
    "createdBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
    "creationDate": 1433147666269,
    "lastModifiedBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
    "lastModifiedDate": 1433147666269
}
```

Validating a Script

To validate a script, perform an HTTP POST using the `/json{/realm}/scripts` endpoint, with an `_action` parameter set to `validate`. Include a JSON representation of the script and the script language, `JAVASCRIPT` or `GROOVY`, in the POST data.

The value for `script` must be in UTF-8 format and then encoded into Base64.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.1" \
--data '{
    "script": "dmFyIGEgPSAxMjM7dmFyIGIgPSA0NTY7Cg==",
    "language": "JAVASCRIPT"
}' \
https://openam.example.com:8443/openam/json/realms/root/realms/myrealm/scripts/?_action=validate
{
    "success": true
}
```

If the script is valid the JSON response contains a `success` key with a value of `true`.

If the script is invalid the JSON response contains a `success` key with a value of `false`, and an indication of the problem and where it occurs, as shown below:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.1" \
--data '{
    "script": "dmFyIGEgPSAxMjM7dmFyIGIgPSA0NTY7ID1WQUxJREFUSU9OIFNIT1VMRCBGQUlMPQo=",
    "language": "JAVASCRIPT"
}' \
https://openam.example.com:8443/openam/json/realms/root/realms/myrealm/scripts/?_action=validate
{
    "success": false,
    "errors": [
        {
            "line": 1,
            "column": 27,
            "message": "syntax error"
        }
    ]
}
```

Creating a Script

To create a script in a realm, perform an HTTP POST using the `/json{/realm}/scripts` endpoint, with an `_action` parameter set to `create`. Include a JSON representation of the script in the POST data.

The value for `script` must be in UTF-8 format and then encoded into Base64.

Note

If the realm is not specified in the URL, AM creates the script in the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.1" \
--data '{
    "name": "MyJavaScript",
    "script": "dmFyIGEgPSAxMjM7CnZhciBiID0gNDU2Ow==",
    "language": "JAVASCRIPT",
    "context": "POLICY_CONDITION",
    "description": "An example script"
}' \
https://openam.example.com:8443/openam/json/realms/root/realms/myrealm/scripts/?_action=create
{
    "_id": "0168d494-015a-420f-ae5a-6a2a5c1126af",
    "name": "MyJavaScript",
    "description": "An example script",
    "script": "dmFyIGEgPSAxMjM7CnZhciBiID0gNDU2Ow==",
    "language": "JAVASCRIPT",
    "context": "POLICY_CONDITION",
    "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
    "creationDate": 1436807766258,
    "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
    "lastModifiedDate": 1436807766258
}
```

Updating a Script

To update an individual script in a realm, perform an HTTP PUT using the `/json{/realm}/scripts` endpoint, specifying the UUID in both the URL and the PUT body. Include a JSON representation of the updated script in the PUT data, alongside the UUID.

> **Note**
>
> If the realm is not specified in the URL, AM uses the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.1" \
--request PUT \
--data '{
    "name": "MyUpdatedJavaScript",
    "script": "dmFyIGEgPSAxMjM7CnZhciBiID0gNDU2Ow==",
    "language": "JAVASCRIPT",
    "context": "POLICY_CONDITION",
    "description": "An updated example script configuration"
}' \
https://openam.example.com:8443/openam/json/realms/root/realms/myrealm/scripts/0168d494-015a-420f-
ae5a-6a2a5c1126af
{
    "_id": "0168d494-015a-420f-ae5a-6a2a5c1126af",
    "name": "MyUpdatedJavaScript",
    "description": "An updated example script configuration",
    "script": "dmFyIGEgPSAxMjM7CnZhciBiID0gNDU2Ow==",
    "language": "JAVASCRIPT",
    "context": "POLICY_CONDITION",
    "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
    "creationDate": 1436807766258,
    "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
    "lastModifiedDate": 1436808364681
}
```

Deleting a Script

To delete an individual script in a realm, perform an HTTP DELETE using the `/json{/realm}/scripts` endpoint, specifying the UUID in the URL.

> **Note**
>
> If the realm is not specified in the URL, AM uses the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--request DELETE \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.1" \
https://openam.example.com:8443/openam/json/realms/root/realms/myrealm/scripts/0168d494-015a-420f-
ae5a-6a2a5c1126af
{}
```

Managing Scripts With the ssoadm Command

Use the **ssoadm** command's **create-sub-cfg**, **get-sub-cfg**, and **delete-sub-cfg** subcommands to manage AM scripts.

Create an AM script as follows:

1. Create a script configuration file, for example `/path/to/myScriptConfigurationFile.txt`, containing the following:

   ```
   script-file=/path/to/myScriptFile.js
   language=JAVASCRIPT ❶
   name=My New Script
   context=AUTHENTICATION_SERVER_SIDE ❷
   ```

   ❶ Possible values for the `language` property are:

   - `JAVASCRIPT`
   - `GROOVY`

   ❷ Possible values for the `context` property are:

   - `POLICY_CONDITION`
   - `AUTHENTICATION_SERVER_SIDE`
   - `AUTHENTICATION_CLIENT_SIDE`
   - `OIDC_CLAIMS`
   - `AUTHENTICATION_TREE_DECISION_NODE`

2. Run the **ssoadm create-sub-cfg** command. The `--datafile` argument references the script configuration file you created in the previous step:

   ```
   $ ssoadm \
     create-sub-cfg \
     --realm /myRealm \
     --adminid amadmin \
     --password-file /tmp/pwd.txt \
     --servicename ScriptingService \
     --subconfigname scriptConfigurations/scriptConfiguration \
     --subconfigid myScriptID \
     --datafile /path/to/myScriptConfigurationFile.txt
   Sub Configuration scriptConfigurations/scriptConfiguration was added to realm /myRealm
   ```

To list the properties of a script, run the **ssoadm get-sub-cfg** command:

```
$ ssoadm \
  get-sub-cfg \
  --realm /myRealm \
  --adminid amadmin \
  --password-file /tmp/pwd.txt \
  --servicename ScriptingService \
  --subconfigname scriptConfigurations/myScriptID
createdBy=
lastModifiedDate=
lastModifiedBy=
name=My New Script
context=AUTHENTICATION_SERVER_SIDE
description=
language=JAVASCRIPT
creationDate=
script=...Script output follows...
```

To delete a script, run the **ssoadm delete-sub-cfg** command:

```
$ ssoadm \
  delete-sub-cfg \
  --realm /myRealm \
  --adminid amadmin \
  --password-file /tmp/pwd.txt \
  --servicename ScriptingService \
  --subconfigname scriptConfigurations/myScriptID
Sub Configuration scriptConfigurations/myScriptID was deleted from realm /myRealm
```

Scripting

amster service name: `Scripting`

Configuration

The following settings appear on the **Configuration** tab:

Default Script Type

The default script context type when creating a new script.

The possible values for this property are:

- `POLICY_CONDITION`. Policy Condition
- `AUTHENTICATION_SERVER_SIDE`. Server-side Authentication
- `AUTHENTICATION_CLIENT_SIDE`. Client-side Authentication
- `OIDC_CLAIMS`. OIDC Claims
- `AUTHENTICATION_TREE_DECISION_NODE`. Decision node script for authentication trees

- `OAUTH2_ACCESS_TOKEN_MODIFICATION`. OAuth2 Access Token Modification

Default value: `POLICY_CONDITION`

amster attribute: `defaultContext`

Secondary Configurations

This service has the following Secondary Configurations.

Engine Configuration

The following properties are available for Scripting Service secondary configuration instances:

Engine Configuration

Configure script engine parameters for running a particular script type in AM.

ssoadm attribute: `engineConfiguration`

To access a secondary configuration instance using the **ssoadm** command, use: `--subconfigname [primary configuration]/[secondary configuration]` For example:

```
$ ssoadm set-sub-cfg \
  --adminid amAdmin \
  --password-file admin_pwd_file \
  --servicename ScriptingService \
  --subconfigname OIDC_CLAIMS/engineConfiguration \
  --operation set \
  --attributevalues maxThreads=300 queueSize=-1
```

> **Note**
>
> Supports server-side scripts only. AM cannot configure engine settings for client-side scripts.

The configurable engine settings are as follows:

Server-side Script Timeout

The maximum execution time any individual script should take on the server (in seconds). AM terminates scripts which take longer to run than this value.

ssoadm attribute: `serverTimeout`

Core thread pool size

The initial number of threads in the thread pool from which scripts operate. AM will ensure the pool contains at least this many threads.

ssoadm attribute: `coreThreads`

Maximum thread pool size

The maximum number of threads in the thread pool from which scripts operate. If no free thread is available in the pool, AM creates new threads in the pool for script execution up to the configured maximum. It is recommended to set the maximum number of threads to 300.

ssoadm attribute: `maxThreads`

Thread pool queue size

Specifies the number of threads to use for buffering script execution requests when the maximum thread pool size is reached.

For short, CPU-bound scripts, consider a small pool size and larger queue length. For I/O-bound scripts, for example, REST calls, consider a larger maximum pool size and a smaller queue.

Not hot-swappable: restart server for changes to take effect.

ssoadm attribute: `queueSize`

Thread idle timeout (seconds)

Length of time (in seconds) for a thread to be idle before AM terminates created threads. If the current pool size contains the number of threads set in `Core thread pool size` idle threads will not be terminated, to maintain the initial pool size.

ssoadm attribute: `idleTimeout`

Java class whitelist

Specifies the list of class-name patterns allowed to be invoked by the script. Every class accessed by the script must match at least one of these patterns.

You can specify the class name as-is or use a regular expression.

ssoadm attribute: `whiteList`

Java class blacklist

Specifies the list of class-name patterns that are NOT allowed to be invoked by the script. The blacklist is applied AFTER the whitelist to exclude those classes - access to a class specified in both the whitelist and the blacklist will be denied.

You can specify the class name to exclude as-is or use a regular expression.

ssoadm attribute: `blackList`

Use system SecurityManager

If enabled, AM will make a call to `System.getSecurityManager().checkPackageAccess(...)` for each class that is accessed. The method throws `SecurityException` if the calling thread is not allowed to access the package.

> **Note**
>
> This feature only takes effect if the security manager is enabled for the JVM.

ssoadm attribute: `useSecurityManager`

Scripting languages

Select the languages available for scripts on the chosen type. Either `GROOVY` or `JAVASCRIPT`.

ssoadm attribute: `languages`

Default Script

The source code that is presented as the default when creating a new script of this type.

ssoadm attribute: `defaultScript`

Appendix C. Getting Support

ForgeRock provides support services, professional services, training through ForgeRock University, and partner services to assist you in setting up and maintaining your deployments. For a general overview of these services, see https://www.forgerock.com.

ForgeRock has staff members around the globe who support our international customers and partners. For details on ForgeRock's support offering, including support plans and service level agreements (SLAs), visit https://www.forgerock.com/support.

ForgeRock publishes comprehensive documentation online:

- The ForgeRock Knowledge Base offers a large and increasing number of up-to-date, practical articles that help you deploy and manage ForgeRock software.

  While many articles are visible to community members, ForgeRock customers have access to much more, including advanced information for customers using ForgeRock software in a mission-critical capacity.

- ForgeRock product documentation, such as this document, aims to be technically accurate and complete with respect to the software documented. It is visible to everyone and covers all product features and examples of how to use them.

Glossary

Access control

Control to grant or to deny access to a resource.

Account lockout

The act of making an account temporarily or permanently inactive after successive authentication failures.

Actions

Defined as part of policies, these verbs indicate what authorized identities can do to resources.

Advice

In the context of a policy decision denying access, a hint to the policy enforcement point about remedial action to take that could result in a decision allowing access.

Agent administrator

User having privileges only to read and write agent profile configuration information, typically created to delegate agent profile creation to the user installing a web or Java agent.

Agent authenticator

Entity with read-only access to multiple agent profiles defined in the same realm; allows an agent to read web service profiles.

Application

In general terms, a service exposing protected resources.

In the context of AM policies, the application is a template that constrains the policies that govern access to protected resources. An application can have zero or more policies.

Application type

Application types act as templates for creating policy applications.

Application types define a preset list of actions and functional logic, such as policy lookup and resource comparator logic.

Application types also define the internal normalization, indexing logic, and comparator logic for applications.

Attribute-based access control (ABAC)

Access control that is based on attributes of a user, such as how old a user is or whether the user is a paying customer.

Authentication

The act of confirming the identity of a principal.

Authentication chaining

A series of authentication modules configured together which a principal must negotiate as configured in order to authenticate successfully.

Authentication level

Positive integer associated with an authentication module, usually used to require success with more stringent authentication measures when requesting resources requiring special protection.

Authentication module

AM authentication unit that handles one way of obtaining and verifying credentials.

Authorization

The act of determining whether to grant or to deny a principal access to a resource.

Authorization Server

In OAuth 2.0, issues access tokens to the client after authenticating a resource owner and confirming that the owner authorizes the client to access the protected resource. AM can play this role in the OAuth 2.0 authorization framework.

Auto-federation

Arrangement to federate a principal's identity automatically based on a common attribute value shared across the principal's profiles at different providers.

Bulk federation

Batch job permanently federating user profiles between a service provider and an identity provider based on a list of matched user identifiers that exist on both providers.

Circle of trust

Group of providers, including at least one identity provider, who have agreed to trust each other to participate in a SAML v2.0 provider federation.

Client

In OAuth 2.0, requests protected web resources on behalf of the resource owner given the owner's authorization. AM can play this role in the OAuth 2.0 authorization framework.

Client-based OAuth 2.0 tokens

After a successful OAuth 2.0 grant flow, AM returns a token to the client. This differs from CTS-based OAuth 2.0 tokens, where AM returns a *reference* to token to the client.

Client-based sessions

AM sessions for which AM returns session state to the client after each request, and require it to be passed in with the subsequent request. For browser-based clients, AM sets a cookie in the browser that contains the session information.

For browser-based clients, AM sets a cookie in the browser that contains the session state. When the browser transmits the cookie back to AM, AM decodes the session state from the cookie.

Conditions

Defined as part of policies, these determine the circumstances under which which a policy applies.

Environmental conditions reflect circumstances like the client IP address, time of day, how the subject authenticated, or the authentication level achieved.

Subject conditions reflect characteristics of the subject like whether the subject authenticated, the identity of the subject, or claims in the subject's JWT.

Configuration datastore

LDAP directory service holding AM configuration data.

Cross-domain single sign-on (CDSSO)

AM capability allowing single sign-on across different DNS domains.

CTS-based OAuth 2.0 tokens

After a successful OAuth 2.0 grant flow, AM returns a *reference* to the token to the client, rather than the token itself. This differs from client-based OAuth 2.0 tokens, where AM returns the entire token to the client.

CTS-based sessions

AM sessions that reside in the Core Token Service's token store. CTS-based sessions might also be cached in memory on one or more AM servers. AM tracks these sessions in order to handle events like logout and timeout, to permit session constraints, and to notify applications involved in SSO when a session ends.

Delegation

Granting users administrative privileges with AM.

Entitlement

Decision that defines which resource names can and cannot be accessed for a given identity in the context of a particular application, which actions are allowed and which are denied, and any related advice and attributes.

Extended metadata

Federation configuration information specific to AM.

Extensible Access Control Markup Language (XACML)

Standard, XML-based access control policy language, including a processing model for making authorization decisions based on policies.

Federation

Standardized means for aggregating identities, sharing authentication and authorization data information between trusted providers, and allowing principals to access services across different providers without authenticating repeatedly.

Fedlet
Service provider application capable of participating in a circle of trust and allowing federation without installing all of AM on the service provider side; AM lets you create Java Fedlets.

Hot swappable
Refers to configuration properties for which changes can take effect without restarting the container where AM runs.

Identity
Set of data that uniquely describes a person or a thing such as a device or an application.

Identity federation
Linking of a principal's identity across multiple providers.

Identity provider (IdP)
Entity that produces assertions about a principal (such as how and when a principal authenticated, or that the principal's profile has a specified attribute value).

Identity repository
Data store holding user profiles and group information; different identity repositories can be defined for different realms.

Java agent
Java web application installed in a web container that acts as a policy enforcement point, filtering requests to other applications in the container with policies based on application resource URLs.

Metadata
Federation configuration information for a provider.

Policy
Set of rules that define who is granted access to a protected resource when, how, and under what conditions.

Policy agent
Java, web, or custom agent that intercepts requests for resources, directs principals to AM for authentication, and enforces policy decisions from AM.

Policy Administration Point (PAP)
Entity that manages and stores policy definitions.

Policy Decision Point (PDP)
Entity that evaluates access rights and then issues authorization decisions.

Policy Enforcement Point (PEP)
Entity that intercepts a request for a resource and then enforces policy decisions from a PDP.

Policy Information Point (PIP)
Entity that provides extra information, such as user profile attributes that a PDP needs in order to make a decision.

Principal
Represents an entity that has been authenticated (such as a user, a device, or an application), and thus is distinguished from other entities.

When a Subject successfully authenticates, AM associates the Subject with the Principal.

Privilege

In the context of delegated administration, a set of administrative tasks that can be performed by specified identities in a given realm.

Provider federation

Agreement among providers to participate in a circle of trust.

Realm

AM unit for organizing configuration and identity information.

Realms can be used for example when different parts of an organization have different applications and identity stores, and when different organizations use the same AM deployment.

Administrators can delegate realm administration. The administrator assigns administrative privileges to users, allowing them to perform administrative tasks within the realm.

Resource

Something a user can access over the network such as a web page.

Defined as part of policies, these can include wildcards in order to match multiple actual resources.

Resource owner

In OAuth 2.0, entity who can authorize access to protected web resources, such as an end user.

Resource server

In OAuth 2.0, server hosting protected web resources, capable of handling access tokens to respond to requests for such resources.

Response attributes

Defined as part of policies, these allow AM to return additional information in the form of "attributes" with the response to a policy decision.

Role based access control (RBAC)

Access control that is based on whether a user has been granted a set of permissions (a role).

Security Assertion Markup Language (SAML)

Standard, XML-based language for exchanging authentication and authorization data between identity providers and service providers.

Service provider (SP)

Entity that consumes assertions about a principal (and provides a service that the principal is trying to access).

Authentication Session

The interval while the user or entity is authenticating to AM.

Session

The interval that starts after the user has authenticated and ends when the user logs out, or when their session is terminated. For browser-based clients, AM manages user sessions across one or more applications by setting a session cookie. See also CTS-based sessions and Client-based sessions.

Session high availability

Capability that lets any AM server in a clustered deployment access shared, persistent information about users' sessions from the CTS token store. The user does not need to log in again unless the entire deployment goes down.

Session token

Unique identifier issued by AM after successful authentication. For a CTS-based sessions, the session token is used to track a principal's session.

Single log out (SLO)

Capability allowing a principal to end a session once, thereby ending her session across multiple applications.

Single sign-on (SSO)

Capability allowing a principal to authenticate once and gain access to multiple applications without authenticating again.

Site

Group of AM servers configured the same way, accessed through a load balancer layer. The load balancer handles failover to provide service-level availability.

The load balancer can also be used to protect AM services.

Standard metadata

Standard federation configuration information that you can share with other access management software.

Stateless Service

Stateless services do not store any data locally to the service. When the service requires data to perform any action, it requests it from a data store. For example, a stateless authentication service stores session state for logged-in users in a database. This way, any server in the deployment can recover the session from the database and service requests for any user.

All AM services are stateless unless otherwise specified. See also Client-based sessions and CTS-based sessions.

Subject

Entity that requests access to a resource

When an identity successfully authenticates, AM associates the identity with the Principal that distinguishes it from other identities. An identity can be associated with multiple principals.

Identity store

Data storage service holding principals' profiles; underlying storage can be an LDAP directory service or a custom `IdRepo` implementation.

Web Agent

Native library installed in a web server that acts as a policy enforcement point with policies based on web page URLs.