Authentication and SSO

These topics cover concepts, implementation procedures, and customization techniques for working with the authentication and single sign-on (SSO) features of ForgeRock Access Management.

Configure AM for authentication

Learn about AM's authentication mechanisms.

Multi-factor authentication

Require that users provide multiple forms of identification when logging in to services.

Single sign-on

Enable single sign-on (SSO) so that users can log in once with a single set of credentials.

Social authentication

Allow users to authenticate to your services by using third-party identity providers.

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.

Introduction to authentication

Authentication is the act of confirming a user's identity, for example, by providing a set of credentials.

In access management, authentication is tightly coupled with *authorization*. Usually, it is important to confirm that a user is who they say they are, *and* to ensure that they can access only a subset of information.

Consider a user who wants to access an online shop. As the owner of the shop, you want to ensure the user identity is confirmed (since it is tied to their shipping and email addresses and payment information) and you also want to ensure that they can only access their own information.

With AM, you can deploy a ForgeRock web agent on the web server hosting the online shop. 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.

In the same way, you can also 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.

AM supports *authentication modules and chains* and *authentication nodes and trees* to implement authentication.

IMPORTANT

Authentication nodes and trees are replacing authentication modules and chains. We recommend that you implement nodes and trees when possible.

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 access to resources. To learn more about sessions, see Sessions.

Multi-factor authentication

Multi-factor authentication (MFA) is an authentication technique that requires users to provide multiple forms of identification when logging in to 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 tree to capture the user's username and password and to create one-time passwords.
2. An end user authenticates to AM using that authentication tree.
3. AM prompts the user to enter the username and password—the first factor in multi-factor authentication.
4. If the user ID and password were correct, AM sends the user an email with a one-time password.
5. The user provides the one-time password to AM to successfully complete authentication—the second factor in multi-factor authentication.

AM supports the following multi-factor authentication protocols:

- MFA: Open AuTHentication (OATH) to enable one-time password authentication.
- MFA: Push authentication to receive push notifications in a device as part of the authentication process.
- MFA: Web authentication (WebAuthn) to enable authentication using an authenticator device, such as a fingerprint scanner.

Configure AM for authentication

AM provides the following features to authenticate users:

Authentication nodes and trees
: AM provides a large variety of authentication nodes, and lets you develop custom nodes, based on your authentication requirements. You connect these nodes to create a *tree* that guides users through the authentication process.

Authentication modules and chains
: AM provides a number of authentication modules to handle different methods of authenticating users. The modules can be *chained* together to provide multiple authentication mechanisms. A user's credentials must be evaluated by one module before control passes to the next module in the chain.

IMPORTANT

Authentication nodes and trees are replacing authentication modules and chains. If your deployment uses modules and chains, you should consider moving to nodes and trees when possible.

The authentication process is extremely flexible, and can be adapted to suit your specific deployment. Although the number of choices can seem daunting, once you understand the basic process, you will be able to configure an authentication path to protect access to most applications in your organization.

Authentication is configured per realm. When a new realm is created, it inherits the authentication configuration of the parent realm. This can save time, especially if you are configuring several subrealms.

The following table summarizes the high-level tasks required to configure authentication in a realm:

Task	Resources
Configure the required authentication mechanisms You need to decide how your users are going to log in. For example, you may require your users to provide multiple credentials, or to log in using third-party identity providers, such as Facebook or Google.	• Authentication nodes and trees • Authentication modules and chains
Configure the realm defaults for authentication Authentication chains and trees use several defaults that are configured at realm level. Review and configure them to suit your environment.	• Configure realm authentication properties
Configure the success and failure URLs for the realm By default, AM redirects users to the UI after successful authentication. No failure URL is defined by default.	• Success and failure redirection URLs
Configure an identity store in your realm. The identity store you configure in the realm should contain those users that would log in to the realm.	• Identity stores

Authentication nodes and trees

Authentication trees (also referred to as Intelligent Authentication) provide fine-grained authentication by allowing multiple paths and decision points throughout the authentication flow. Use them to build complex authorization scenarios, while offering a streamlined login experience to users.

Authentication trees are made up of authentication nodes, which define actions taken during authentication. Each node performs a single task during authentication, for example, collecting a username or making a simple decision based on a cookie.

Nodes can have multiple outcomes rather than just success or failure. This allows you to create complex yet customer-friendly authentication experiences by linking nodes together, creating loops, branching the tree for different authentication scenarios, and nesting nodes within a tree:

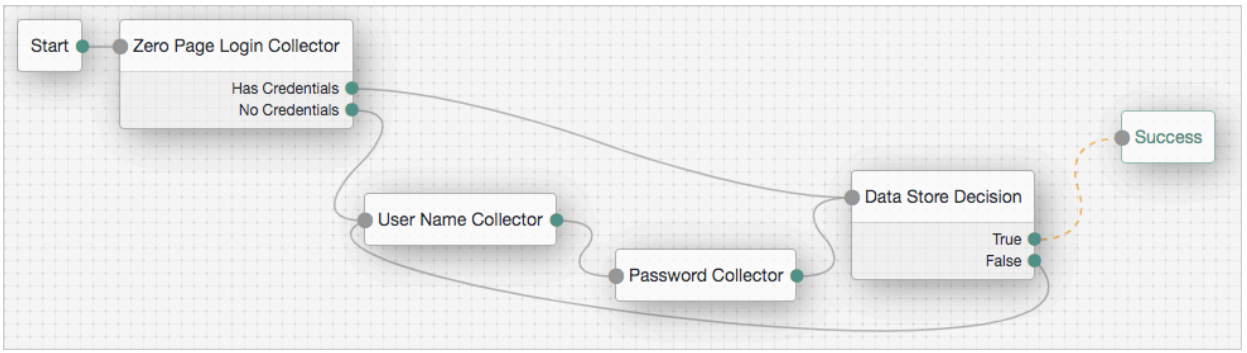


Figure 1. Example authentication tree

To further control the authentication process, you can assign authentication levels to branches on a tree, with higher levels being used typically to allow access to more restricted resources.

Authentication trees differ in the following ways from 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 Configure authentication webhooks. Note that post-

authentication plugins do not get triggered when authenticating to a tree, only to a chain.

Authentication levels for trees

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*. The authentication level is often used as a measure of the strength of the authentication performed. For example, simple username and password may be assigned a low authentication level, whereas multi-factor with Push and webAuthn, a high one.

Authorization policies may require a particular authentication level to access protected resources. When an authenticated user tries to access a protected resource without satisfying the authentication level requirement, AM 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, refer to Session upgrade

AM provides the following nodes to manage authentication levels:

- The `Authentication Level Decision` node, that checks that the current authentication level is equal or greater than the one specified in the node.
- The `Modify Authentication Level` node, that can raise or lower the authentication level.

Position these nodes to alter the authentication level depending on the route take through the authentication tree.

Account lockout for trees

You should limit the number of attempts a user can make at authenticating with credentials. Limiting the number of attempts helps to prevent password-guessing and brute-force attacks.

By default, authentication trees support account lockout and provide nodes for checking and changing a user's status:

Account Active Decision node
: Use this node to determine if an account is *active* or *inactive* (locked).

Account Lockout node
: Use this node to change the user's status to *active* or *inactive* (locked).

NOTE

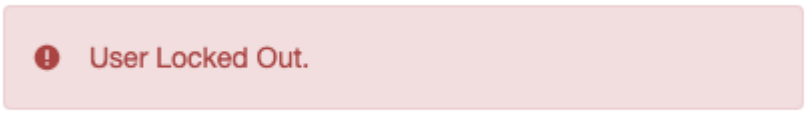
NOTE

When setting an account to active, the node also resets the *failed attempts* and *lockout duration* counters.

In addition to the lockout-specific nodes above, the *Success* and *Failure* nodes include account lockout functionality, when lockout is enabled in a realm, as follows:

Success node

- Checks the **User Status** property of the user profile, when reached, and fails the authentication with an error message, if the account is marked as `Inactive`:

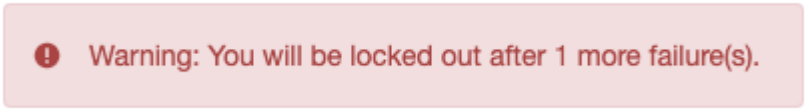
  The error message is returned in the JSON response if authenticating to the tree by using REST:

  ```
  {
      "code":401,
      "reason":"Unauthorized",
      "message":"User Locked Out."
  }
  ```

- Resets the failure count in the user profile, when reached, if the **User Status** property is set to `Active`.

Failure node

- Checks the invalid attempts property of the user profile, and returns a warning message if the number of failed attempts is equal to or greater than the configured **Warn User After N Failures** value in the realm:

  The error message is returned in the JSON response if authenticating to the tree by using REST:

  ```
  {
      "code":401,
      "reason":"Unauthorized",
      "message":"Warning: You will be locked out after 1 more
  failure(s)."
  }
  ```

- Increments the failure count in the user profile, when reached.

- Returns an error message if the account is marked as `Inactive`:

The error message is returned in the JSON response if authenticating to the tree by using REST:

```
{
    "code":401,
    "reason":"Unauthorized",
    "message":"User Locked Out."
}
```

For information on configuring account lockout in a realm, refer to Account lockout.

Specify IDM identity resources in trees

When running AM as part of an integrated platform with IDM, trees configured to use the platform need to identify the type of identity resource or object the tree is working with. To do this, use the `identityResource` configuration property. If the property is not included in the tree configuration, it defaults to `managed/user`.

To update `identityResource` on a tree, use the REST API to update the tree:

```
$ curl \
 --request PUT \
 --header 'Accept-API-Version: protocol=2.1,resource=1.0' \
 --header 'Accept: application/json' \
 --header 'If-None-Match: *' \
 --header 'Content-Type: application/json' \
 --header 'Cookie: <omitted for length>' \
 --data '{
   "entryNodeId":"e301438c-0bd0-429c-ab0c-66126501069a",
   "nodes":{},
   "staticNodes":{},
   "description":"Example tree description",
   "identityResource":"managed/newObjectType"
 }' \
 "https://default.iam.example.com/am/json/realms/root/realm-
config/authentication/authenticationtrees/trees/ExampleTree"
```

In the above example, the tree `ExampleTree` has no nodes added to it yet. It includes the `identityResource` property, set to use a managed object in IDM called

`newObjectType`.

Because this is a `PUT` request, you must include the entire tree as part of the request. For more information about using the REST API, refer to REST in AM.

Configure authentication trees

The following table summarizes the high-level tasks required to configure authentication trees:

Task	Resources
Design your user authentication journey Authentication trees are very flexible. For example, the same tree can branch for different use cases, or users can be forced to loop though branches until they are able to present the required credentials. It is easy to create a massive tree that is difficult to understand, read, and maintain in the UI. For this reason, AM allows you to nest trees within trees. The best way to tackle the design decision is to write down a list of required steps users would need to take to log in to your environment, and then check the list of nodes available in AM. TIP Evaluation installs of AM that use the 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 in instances of AM that use an external configuration store, or if you are upgrading an existing instance of AM. To obtain a copy of the sample trees that you can import into your instance, see How do I access and build the sample code provided for PingAM? in the *Knowledge Base*. For information on importing the sample tree JSON files by using Amster, see Import configuration data in the Amster documentation.	• Authentication nodes configuration reference, for a list of nodes delivered with AM. • Marketplace website, for additional nodes certified by Ping Identity or our partners. • Multi-factor authentication, to understand how multi-factor authentication works with trees. • Social authentication, to understand how social authentication works with trees.

Task	Resources
Decide if you need custom authentication nodes and webhooks If the nodes available in AM or in the ForgeRock Marketplace do not suit your needs, you can build your own nodes. In the same way, you can create custom webhooks for nodes that need them.	• Node development. • Create post-authentication hooks for trees.
Configure your authentication trees Use the authentication tree designer to put together your trees quickly.	• Create an authentication tree.
Configure webhooks, if required If you have configured the `Register Logout Webhook` node, configure its webhook.	• Configure authentication webhooks.

Create an authentication tree

1. In the AM admin UI, go to **Realms** > ***Realm Name*** > **Authentication** > **Trees** and click **Create Tree**.
2. Enter a tree name, for example `myAuthTree` , and click **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:

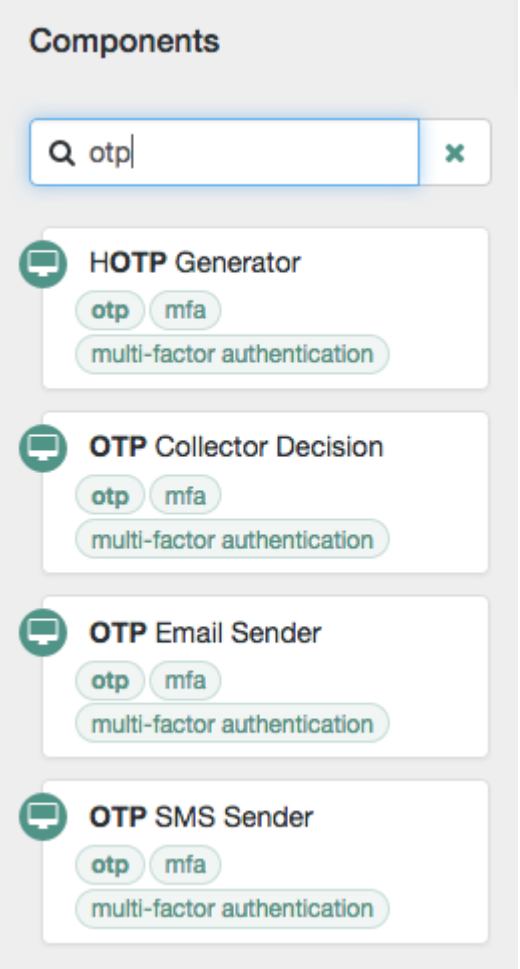
   Authentication tree designer toolbar

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.

3. 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.

   The list of authentication nodes is split into a number of categories. Click the category title to expand and collapse the categories.

   Use the filter text field to restrict the list of authentication nodes, which will match on the nodes' name, and any tags applied to the node:



4. Configure the node properties by using the right-hand panel.

   For more information on the available properties for each node, see Authentication nodes configuration reference.

5. Connect the node to the tree as follows:

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

   b. 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.

6. To change a connection, select and drag the green connector in the connection and drop it onto the new location.
7. Continue adding, connecting and removing nodes until the tree is complete, then click **Save**.
8. Test your authentication tree by navigating to a URL similar to the following: `https://openam.example.com:8443/openam/XUI/?realm=/alpha&service=myAuthTree#login`

Enable and disable an authentication tree

Custom authentication trees are *enabled* by default, when they are saved. For security purposes, you can disable custom authentication trees during development and testing, to prevent accidentally allowing access through these trees. Rather than having unused authentication trees enabled, you should disable the default authentication trees until you need them.

When a user attempts to authenticate through a disabled tree, AM returns a `Tree does not exist` error.

To enable or disable an authentication tree, set the `enabled` flag in the tree configuration:

Over REST, send a PUT request to update the tree configuration. You must specify the tree ID and the nodes in the tree. This example disables the `myAuthTree` created previously:

```
$ curl \
--header "Content-Type: application/json" \
--header 'Cookie: iPlanetDirectoryPro=<cookie>' \
--header 'accept-api-version: protocol=2.1,resource=1.0' \
--header "If-Match: *" \
--request PUT \
--data '
  {
  "entryNodeId": "c11e9cf8-ef48-4740-876f-6300e2f46aef",
  "nodes": {
    "c11e9cf8-ef48-4740-876f-6300e2f46aef": {
      "displayName": "Page Node",
```

```
      "nodeType": "PageNode",
      "x": 147,
      "y": 25,
      "connections": {
        "outcome": "15839e1c-5085-4f58-bc94-c4cc848a0ae8"
      }
    },
    "15839e1c-5085-4f58-bc94-c4cc848a0ae8": {
      "displayName": "Data Store Decision",
      "nodeType": "DataStoreDecisionNode",
      "x": 349,
      "y": 25,
      "connections": {
        "true": "70e691a5-1e33-4ac3-a356-e7b6d60d92e0",
        "false": "e301438c-0bd0-429c-ab0c-66126501069a"
      }
    }
  },
  "enabled": false
}' \
"https://openam.example.com:8443/openam/json/realms/root/realms/alpha/realm-config/authentication/authenticationtrees/trees/myAuthTree"
{
  "_id": "myAuthTree",
  "_rev": "2070284866",
  "uiConfig": {},
  "entryNodeId": "c11e9cf8-ef48-4740-876f-6300e2f46aef",
  "nodes": {
    "c11e9cf8-ef48-4740-876f-6300e2f46aef": {
      "displayName": "Page Node",
      "nodeType": "PageNode",
      "x": 147,
      "y": 25,
      "connections": {
        "outcome": "15839e1c-5085-4f58-bc94-c4cc848a0ae8"
      }
    },
    "15839e1c-5085-4f58-bc94-c4cc848a0ae8": {
      "displayName": "Data Store Decision",
      "nodeType": "DataStoreDecisionNode",
      "x": 349,
      "y": 25,
      "connections": {
        "true": "70e691a5-1e33-4ac3-a356-e7b6d60d92e0",
```

```
        "false": "e301438c-0bd0-429c-ab0c-66126501069a"
      }
    }
  },
  "staticNodes": {
    "startNode": {
      "x": 50,
      "y": 25
    },
    "70e691a5-1e33-4ac3-a356-e7b6d60d92e0": {
      "x": 570,
      "y": 30
    },
    "e301438c-0bd0-429c-ab0c-66126501069a": {
      "x": 573,
      "y": 107
    }
  },
  "enabled": false
}
```

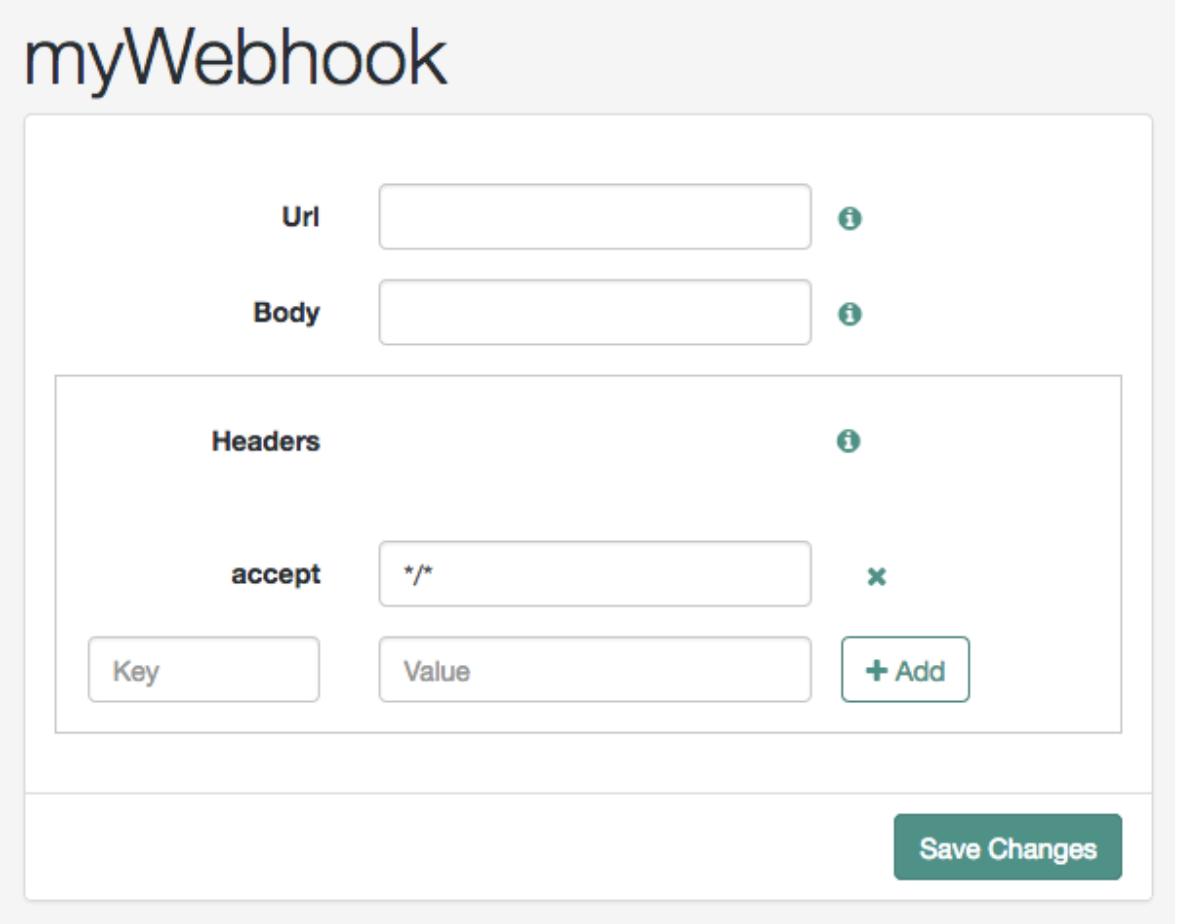
Configure authentication webhooks

Webhooks 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 Register Logout Webhook node.

Create a webhook

1. In the AM admin UI, go to **Realms** > ***Realm Name*** > **Authentication** > **Webhooks**.
   a. To create a new webhook, select **Create Webhook**, specify a webhook name, and click **Create**.
   b. To edit an existing webhook, select the name of the webhook.



2. 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 click **Add** (**+**).

   To remove a header, click **Delete** (**✖**).

The fields in a webhook support variables for retrieving values from the user's session after successfully authenticating.

Specify a variable in the following format:

```
${variable_name}
```

To access the type of webhook event, use the `WebhookEventType` parameter key to return one of the following possible values:

- `LOGOUT`
- `UPGRADE`
- `DESTROY`
- `MAX_TIMEOUT`
- `IDLE_TIMEOUT`

For example, to retrieve the event type as a query parameter:
`&event=${WebhookEventType}`

You can use a variable to access custom properties added to the session with the Set Session Properties node as well as the following default session properties:

▼ *Default Session Properties*

Property	Example value	Description
`AMCtxId`	22e73c81-708e-4849-b064-db29b68ef943-105372	The audit ID for the session. This is logged as the `trackingIds` field in AM access audit logs.
`amlbcookie`	01	The cookie that identifies the AM server that generated the session. For environments with multiple AM servers, this can be used for load balancer stickiness.
`authInstant`	2022-02-28T14:06:31Z	The exact time that authentication completed.

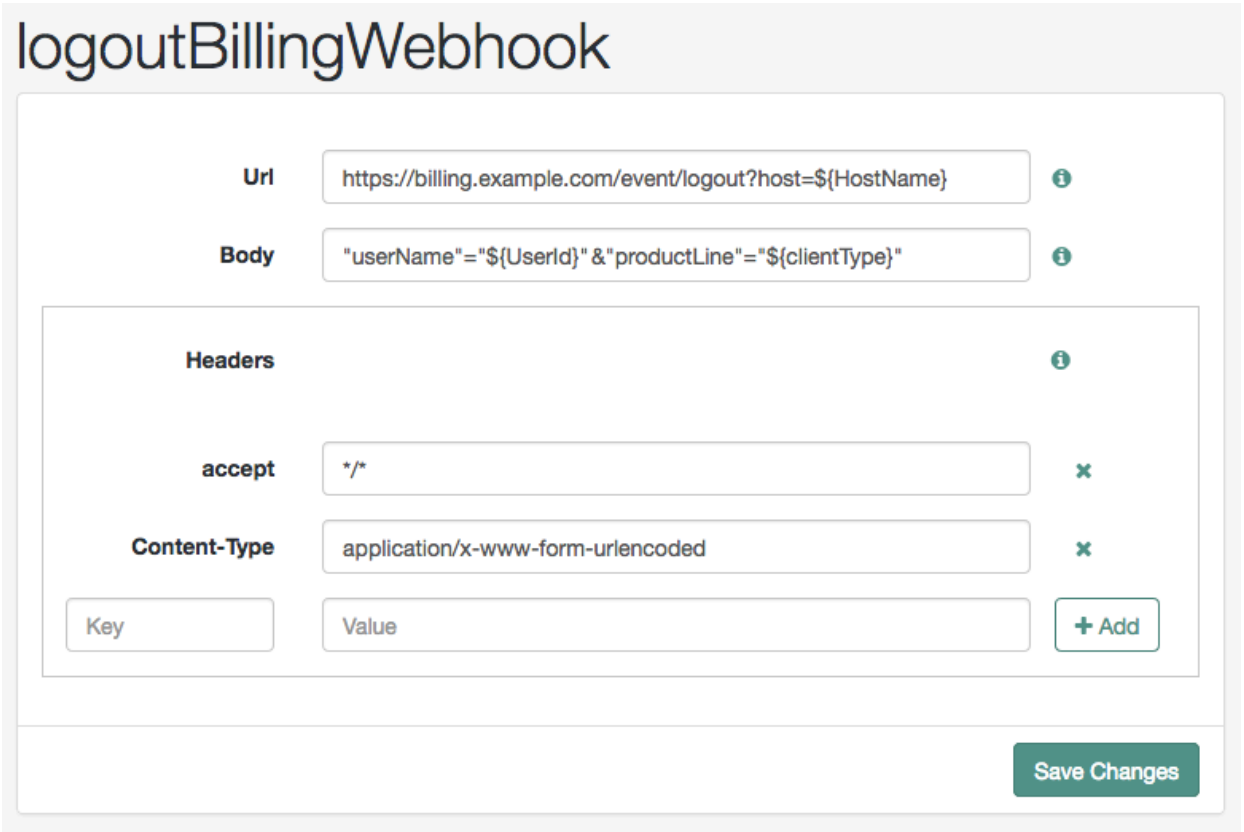
Property	Example value	Description
`AuthLevel`	5	The authentication level of the session, determined by the login mechanism used to create the session. For example, a tree can have an authentication level of 10. Step-up authentication is triggered if an authentication level specified by an agent or policy that is designed to protect a resource, is greater than or equal to the value of the `AuthLevel` session property. For more information, see Session upgrade.
`AuthType`	DataStore	A pipe-separated list of modules to which the user has authenticated. For example, `module1|module2|module3` (*authentication chains only*).
`CharSet`	UTF-8	The character set for the session, set to `UTF-8`.
`clientType`	genericHTML	The type of client, set to `genericHTML`.
`FullLoginURL`	/openam/XUI/?realm=%2Falpha	The full login URL, including query parameters.
`Host`	192.0.2.1	The originating IP address of the authentication request (*authentication trees only*).
`HostName`	192.0.2.1	The host name that was used when the session was authenticated.
`IndexType`	service	Based on the value of the `authIndexValue` query parameter during authentication. Typically, this is set to `service`.
`Locale`	en_US	The session locale.
`loginURL`	/openam/XUI	The base login URL. A subset of `FullLoginURL`.

Property	Example value	Description
`OidcSid`	g0wmSpoAIwH6HAwCnurvRcfYqh4	Unique session ID used by AM to determine whether OIDC ID tokens granted for the same client relate to the same session. This appears when `Enable Session Management` (`storeOpsToken`) is set to true in the OAuth 2.0 provider settings.
`Organization`	o=alpha,ou=services,dc=openam,dc=forgerock,dc=org	The DN of the realm where authentication took place.
`Principal`	id=demo,ou=user,o=alpha,ou=services,dc=openam,dc=forgerock,dc=org	The value of `sun.am.UniversalIdentifier` .
`Principals`	demo	The username of the user. For multiple principals, this can be a pipe-separated list (*authentication chains only*).
`Service`	ldapService	The name of the tree or chain that was used to authenticate this session.
`successURL`	/openam/console	The URL that was redirected to, upon a successful login request.
`sun.am.UniversalIdentifier`	id=demo,ou=user,o=alpha,ou=services,dc=openam,dc=forgerock,dc=org	The DN of the user (username is lowercase).
`UserId`	demo	The `id` value from the `Principal` property.

Property	Example value	Description
`UserProfile`	Required	Can be one of: `Required`, `Create`, `Ignore`, or `CreateWithAlias`. Based on the value of the `dynamicProfileCreation` authentication configuration. Values other than `Ignore` indicates that user profile attributes were mapped based on the `User Attribute Mapping to Session Attribute` setting. See authentication configuration for details. Default: `Required`.
`UserToken`	demo	The username, as defined in the `Principals` property. For authentication chains, this is the last principal value.

Example webhook

The following figure shows an example webhook using variable substitutions:



WARNING

WARNING

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

Customize authentication trees

Your deployment might require customizing standard authentication trees.

For information on customizing authentication nodes, refer to Node development.

Create 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:

```
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) ①
public class UpdatePersistentCookieTreeHook implements TreeHook {
②

    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 ③
    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 { ④
        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;
```

```
    }
}
```

① 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.

NOTE

The node class must invoke ActionBuilder's `addSessionHook` method to specify the treehook class to be run after a successful login.

For example, in the `PersistentCookieDecisionNode.class`:

```
@Override
public Action process(TreeContext context) throws NodeProcessException
{
    ...
    actionBuilder = actionBuilder
        .addSessionHook(UpdatePersistentCookieTreeHook.class, nodeId,
getClass().getSimpleName());
}
```

② The core class must implement the `TreeHook` interface.

For more information, see the TreeHook interface in the *AM Public API Javadoc*.

③ 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*.

④ Creating an `Accept` instance. The main logic of a tree hook is handled by the `Accept` function.

Authentication modules and chains

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.

Criteria	Fail	Pass	Example
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.  Continues chain.	Assigns pass flag.  Continues chain.	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
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:

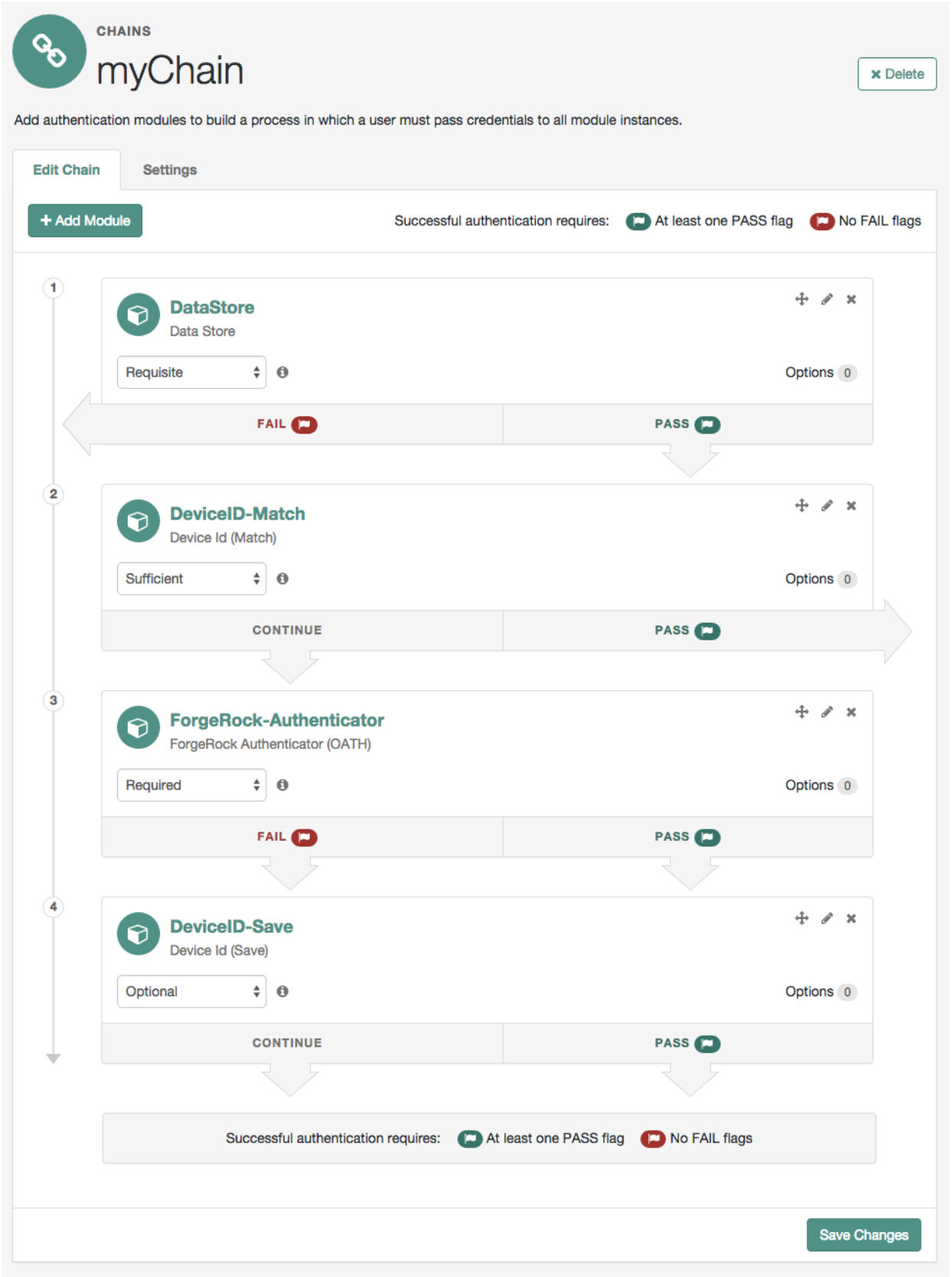


Figure 2. Authentication Chain with Each Criteria

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.

Authentication levels for chains

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. 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 protected resources. When an authenticated user tries to access a protected resource without satisfying the authentication level requirement, AM 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.

Configure authentication chains

The following table summarizes the high-level tasks required to configure authentication chains:

Task	Resources
Design the authentication journey of your users The best way to tackle the design decision is to write down a list of required steps users would need to take to log in to your environment, and then check the list of modules available in AM.	• Authentication modules configuration reference, for a list of modules delivered with AM. • About Multi-Factor Authentication, to understand how multi-factor authentication works with chains. • Social authentication, to understand how social authentication works with chains. NOTE You can find information on implementing authentication using WS-Federation in How do I configure PingAM as an Identity Provider for Microsoft Office 365 and Azure using WS-Federation? in the *Knowledge Base*.
Decide if you need custom authentication modules, server-side Scripts, or post-authentication plugins If the default authentication modules and plugins do not suit your needs, consider coding your own.	• Customize authentication chains.
Configure authentication modules Before setting up your chains, configure the authentication modules you need.	• Configure authentication modules.
Configure authentication chains Use the chain designer to put together your chains quickly.	• Create an authentication chain.
Configure post-authentication plugins Post-authentication plugins allow you to manage session properties and run scripts after authentication, or after the user has logged out.	• Post-authentication plugins.

Configure authentication modules

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

- Under **Configure > Authentication**, you configure default properties for global authentication modules.
- Under **Realms > Realm Name > Authentication v Modules**, you configure modules for your realm.

To configure the authentication modules required by your environment, refer to the following resources:

- MFA: Open AuTHentication (OATH)
- MFA: Push authentication
- MFA: Web authentication (WebAuthn)
- Social authentication

For module reference information, see Authentication modules configuration reference.

Create an authentication chain

After 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.

1. In the AM admin UI, go to **Realms > *Realm Name* > Authentication > Chains**.
2. On the **Authentication Chains** page, click **Add Chain**. Enter a chain name, and click **Create**.

   In the **Edit Chain** dialog, click **Add a Module**.
3. Choose the authentication module in the drop-down list, and then assign appropriate criteria (Optional, Required, Requisite, Sufficient).

   Add as many modules as required.
4. If you need modules in the chain to share user credentials, consider the following available options:

   ▼ *Options to share credentials among modules*

   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`

Enter the key and its value, and click Plus (+). When you finish entering the options, click OK.

▼ *Examples*

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`.

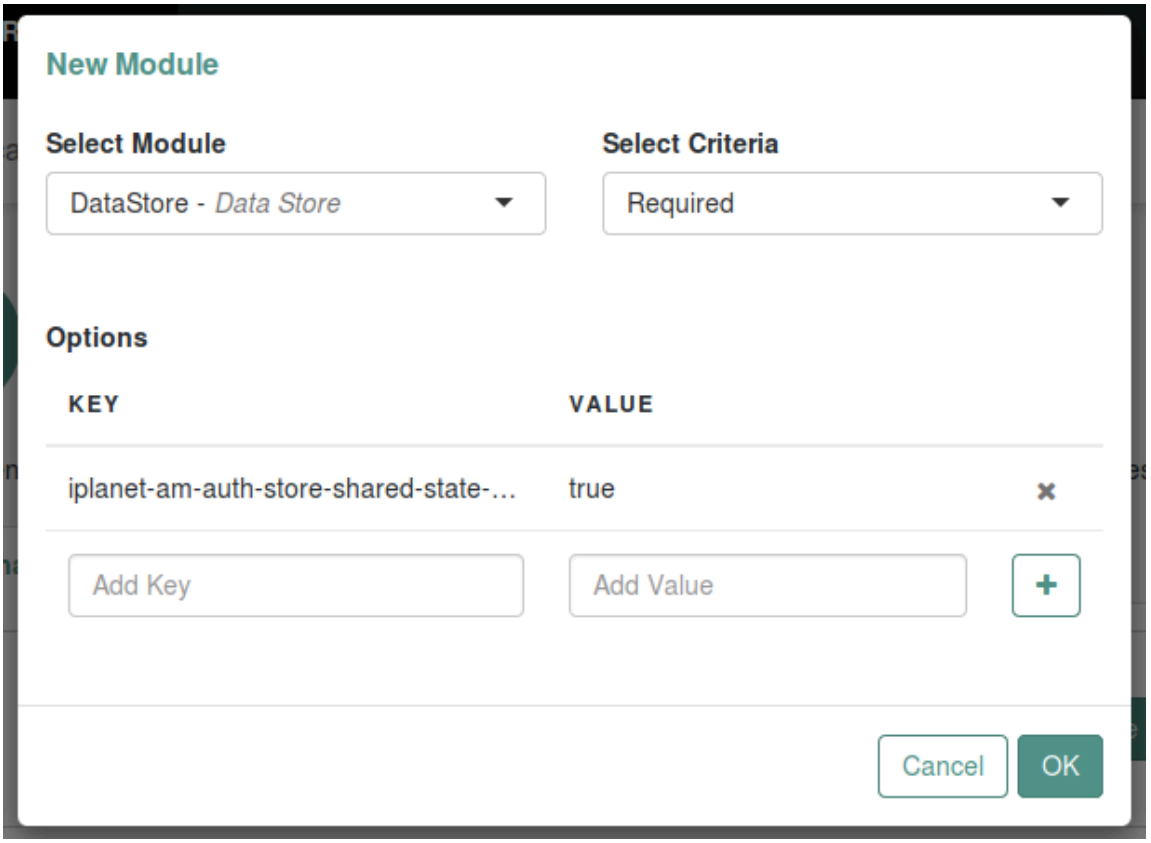


Figure 3. Authentication chain first module

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`.

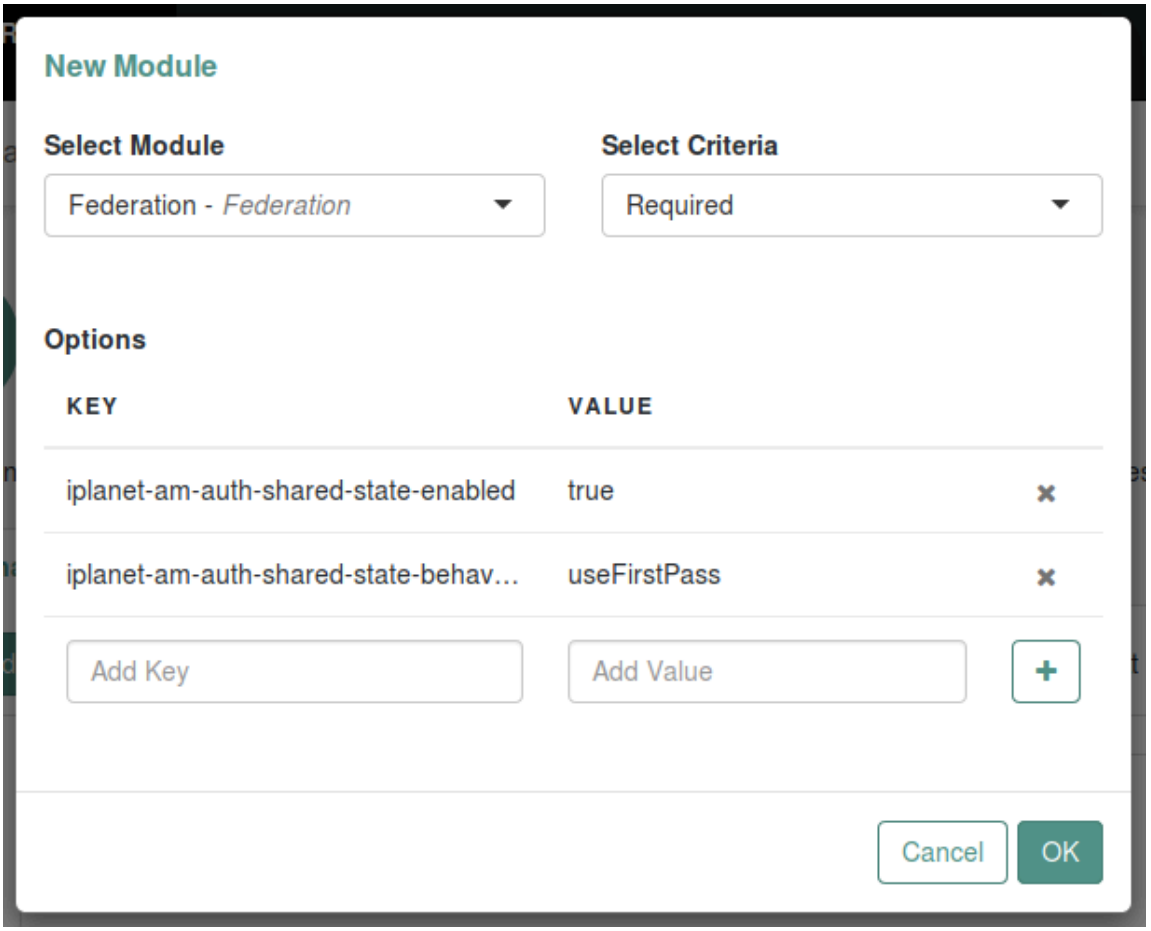


Figure 4. Authentication chain second module

5. On the **Settings** tab, configure where AM redirects the user upon successful and failed authentication, and plug in your post-authentication processing classes as necessary.

   If you configure absolute URLs that are not in the same scheme, FQDN, and port as AM, you must also configure the Success and failure redirection URLs.

6. Save your work.

   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.

Login session timeouts for chains

Login pages have a *session timeout* that specifies the number of minutes before the session times out, if the user has not logged in. The login session timeout has two components:

- The timeout of the specific authentication module.

  The default session timeout for an authentication module is two minutes.

- The overall session timeout, set in **Configure > Server Defaults > Session > Session Limits > Invalidate Session Max Time**.

  The default overall session timeout is three minutes.

You must set the overall session timeout to a value greater than the complete authentication process (including any multi-page authentication processes). If you have chained authentication modules, with different timeouts, you must set the overall session timeout to a value greater than the sum of these timeouts.

Learn more in How do I configure login page session timeouts in PingAM when using authentication modules? in the *Knowledge Base*.

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 admin UI, you add post-authentication plugins to an authentication chain. Go to Realms > *Realm Name* > Authentication > Chains > *Auth Chain Name* > Settings > Post Authentication Processing Class > Class Name.

See Create 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`***
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. (Only one password replay post-authentication plugin class can be active for a given AM deployment.)

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 the property, decrypts it, and replays the password into legacy applications.

To configure password replay for AM and IG, refer to Get login credentials from AM in the ForgeRock Identity Gateway documentation.

*Class name: **`com.sun.identity.authentication.spi.JwtReplayPassword`***
Password replay post-authentication plugin class that uses a JWT-based AES A128CBC-HS256 encryption algorithm. (Only one password replay post-authentication plugin class can be active for a given AM deployment.)

The plugin encrypts the password captured by AM during the authentication process and stores it in a session property. IG looks up 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, refer to Get login credentials from AM in the ForgeRock Identity Gateway documentation.

If necessary, you can also write your own custom post-authentication plugin as described in Create post-authentication plugins for chains.

Customize authentication chains

Your deployment might require customizing standard authentication chain features.

Create 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.

The sample custom authentication module

The sample custom 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 admin UI, 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 PingAM? 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 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
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 Javadoc.

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`.

```
#
# 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 Software, 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 copyright [year] [name of copyright
```

```
owner]".
#
# Copyright 2011-2017 ForgeRock AS. All Rights Reserved
#

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 `SampleAuth.java`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 * 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 Software, 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 copyright [year] [name of copyright
owner]".
 *
 * Copyright 2011-2017 ForgeRock AS. All Rights Reserved
 *
-->
<!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.

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:

```
/*
 * 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 Software, 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 copyright [year] [name of copyright
owner]".
 *
 * Copyright 2011-2022 ForgeRock AS. All Rights Reserved
 */

package org.forgerock.openam.examples;

import java.security.Principal;
import java.util.Map;
import java.util.ResourceBundle;
import java.util.Set;

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 org.slf4j.Logger;
import org.slf4j.LoggerFactory;

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;

/**
 * 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 Logger debug =
LoggerFactory.getLogger(SampleAuth.class);

    // 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 = "Ch4ng31t";

    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, Set<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.debug("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.debug("SampleAuth::process state: {}", state);

        switch (state) {

            case STATE_BEGIN:
                // Intialize Callback list if used in chain with
                // iplanet-am-auth-shared-state-enabled=true
                setForceCallbacksRead(true);
                forceCallbacksInit();

                // 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.debug("SampleAuth::process User '{}' " +
                            "authenticated with success.",
username);
                    return ISAuthConstants.LOGIN_SUCCEED;
                }

                throw new InvalidPasswordException("password is
wrong",
                        USERNAME, isReturningPrincipalAsDn());

            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:

```
/*
 * 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 Software, 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 copyright [year] [name of copyright
owner]".
 *
 * Copyright 2011-2017 ForgeRock AS. All Rights Reserved
 */

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
<code>SampleAuthPrincipal</code>.
     *
     * @return the LDAP username for this
<code>SampleAuthPrincipal</code>
     */
    @Override
    public String getName() {
        return name;
```

```
    }

    /**
     * Return a string representation of this
<code>SampleAuthPrincipal</code>.
     *
     * @return a string representation of this
     *         <code>TestAuthModulePrincipal</code>.
     */
    @Override
    public String toString() {
        return new
StringBuilder().append(this.getClass().getName())
                .append(COLON).append(name).toString();
    }

    /**
     * Compares the specified Object with this
<code>SampleAuthPrincipal</code>
     * for equality. Returns true if the given object is also a
     * <code> SampleAuthPrincipal </code> and the two
SampleAuthPrincipal have
     * the same username.
     *
     * @param o Object to be compared for equality with this
     *          <code>SampleAuthPrincipal</code>.
     * @return true if the specified Object is equal equal to this
     *         <code>SampleAuthPrincipal</code>.
     */
    @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
<code>SampleAuthPrincipal</code>.
     *
     * @return a hash code for this
<code>SampleAuthPrincipal</code>.
     */
    @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 admin UI.

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"?>
<!--
 * 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 Software, 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 copyright [year] [name of copyright
owner]".
 *
 * Copyright 2011-2020 ForgeRock AS. All Rights Reserved
 *
-->
<!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/iPlanetAMAuthSampleA
uthService"
   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 Custom 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 PingAM? 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.

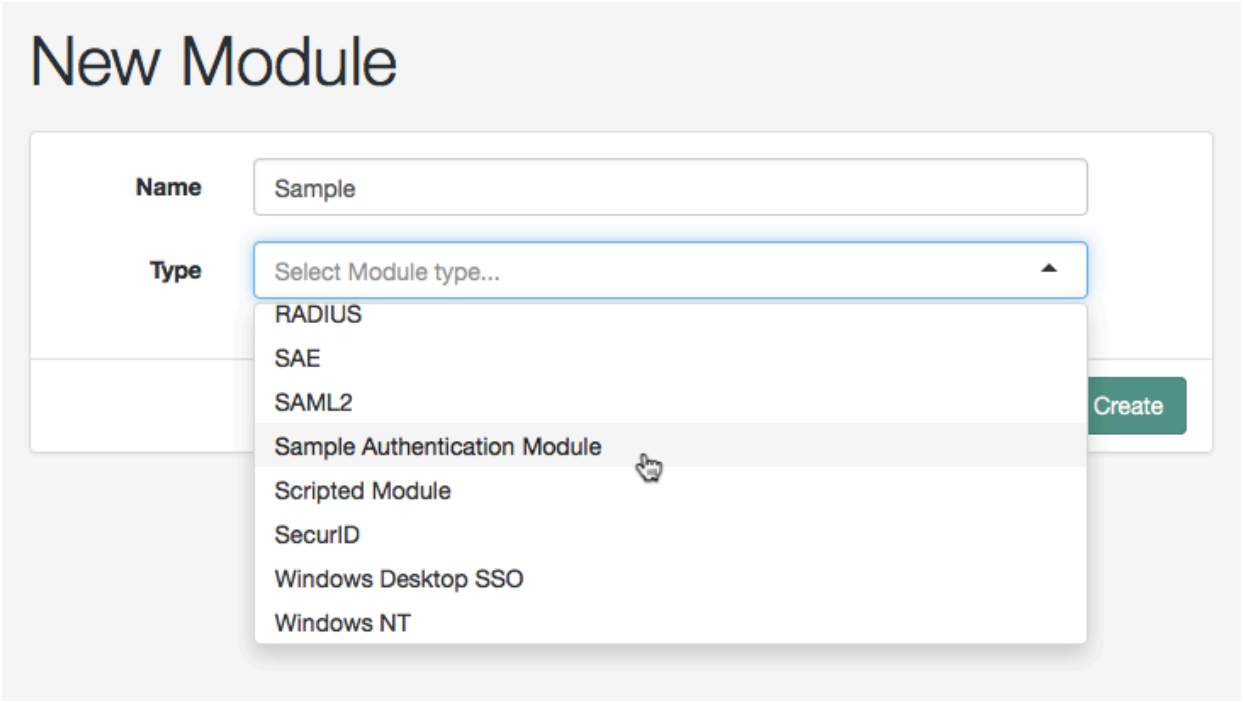
   For example if you deployed AM in Apache Tomcat, then you shut down Tomcat and start it again.

   ```
   $ /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
   ```

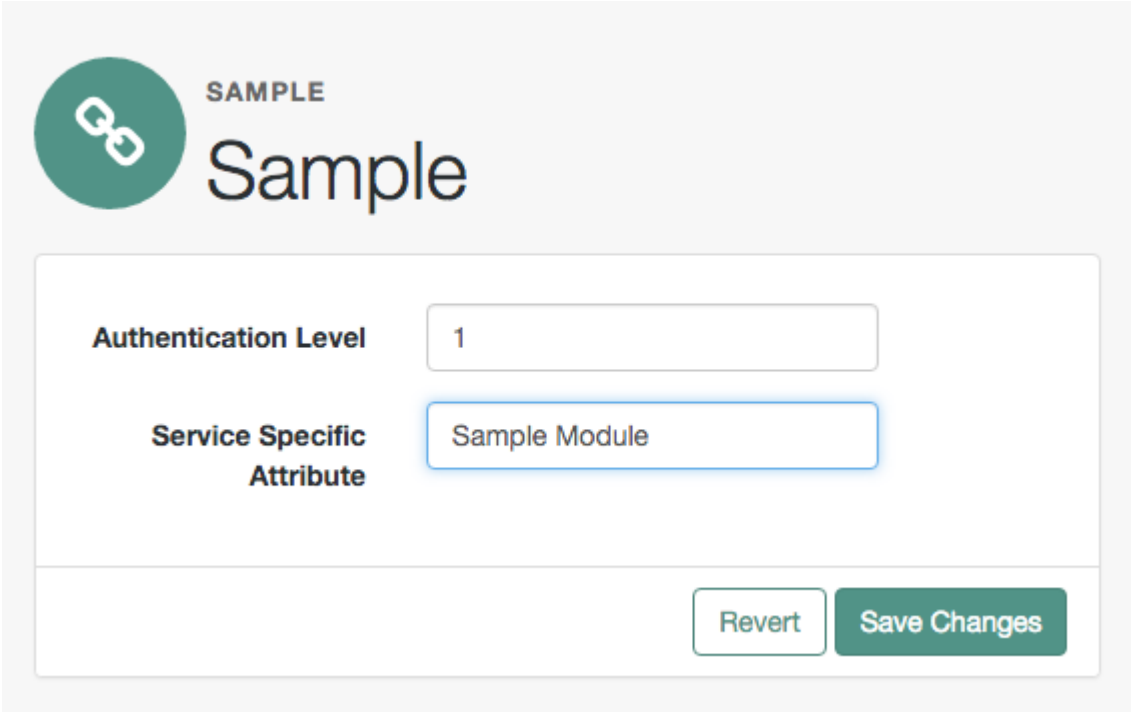
Configure and test the Sample Auth module

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).

In the AM admin UI, go to **Realms** > **Top Level Realm** > **Authentication** > **Modules**. Click Add Module to create an instance of the Sample Authentication Module. Name the module `Sample`.



Click **Create**, and then configure the authentication module as appropriate.



Now that the module is configured, log out of the AM admin UI.

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 `Ch4ng31t`.

After authentication you are redirected to the end user page for the demo user. You can logout of the AM admin UI, and then try to authenticate as the non-existent user `test123` to see what the error handling looks like to the user.

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 admin UI, go to **Realms > Top Level Realm > Scripts**, and click **Scripted Module - Server Side**.

For general information about scripting in AM, see Scripting.

Prepare AM to use server-side authentication scripts

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.

Create a scripted authentication module that uses the default server-side authentication acript

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. Go 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 click **Create**.
5. On the module configuration page:
   - Uncheck the **Client-side Script Enabled** checkbox.
   - From the **Server-side Script** drop-down list, select `Scripted Module - Server Side` .
   - Click **Save Changes**.

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. Go 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 click **Create**.
5. On the **Edit Chain** tab, click **Add a Module**.
6. In the **New Module** dialog box:
   - From the **Select Module** drop-down list, select `DataStore` .
   - From the **Select Criteria** drop-down list, select `Required` .
   - Click **OK**.

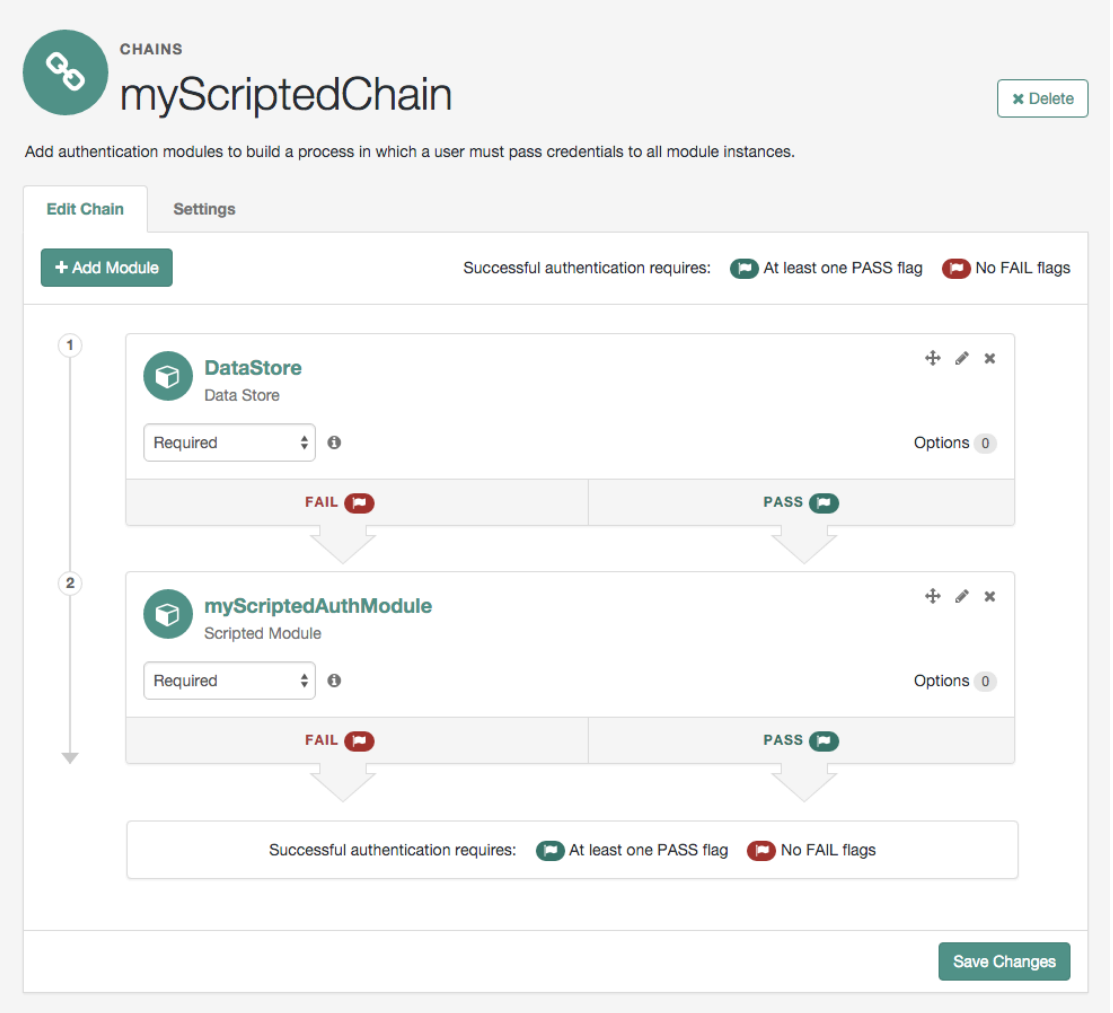
NOTE

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:
    - From the **Select Module** drop-down list, select the Scripted Module from the previous procedure, for example `myScriptedAuthModule` .
    - From the **Select Criteria** drop-down list, select `Required` .
    - Click **OK**.

      The resulting chain resembles the following:



9. On the **Edit Chain** tab, click **Save Changes**.

Add a postal address to the demo user

1. Log in as an AM administrator, for example `amAdmin` .

2. Go 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.

Log in using a chain containing a Scripted Authentication module

1. Log out of AM.
2. In a browser, go 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 `Ch4ng31t`.

   If login is successful, the user profile page is displayed. 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

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.

4. 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:
    - Log out the `demo` user.
    - Log in as an AM administrator, for example `amAdmin` .
    - Go to **Realms > Top Level Realm > Scripts > Scripted Module - Server Side**.
    - In the script, swap the values for `START_TIME` and `END_TIME` , for example:

```
var START_TIME = 17;
var END_TIME   = 9;
```

    - Click **Save**.
    - 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.

Create 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-side sessions:

Cookie size
: You can set an unlimited number of session properties in a post-authentication plugin. When AM creates a client-side 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-side 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-side session cookies, see Session cookies and session security.

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-side 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-side sessions, be sure that you are aware of the measures securing the session contained within the cookie.

For more information about client-side session cookie security, see Configure client-side session security.

Design 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 plugin in the top level realm can have unexpected effects. At the top level realm, AM invokes the post-authentication plugin for all types of authentication during startup, including user logins and internal administrative logins. The best practice is to let end users into subrealms only, and administrators into the top level realm. If you *must* 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.

Realm-level post-authentication plugins are only called when no post-authentication plugin is configured for the authentication chain.

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.

Build 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.

```
/*
 * 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 Software, 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 copyright [year] [name of copyright
owner]".
 *
 * Copyright 2011-2019 ForgeRock AS. All Rights Reserved
 */

package com.forgerock.openam.examples;

import java.util.Map;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.iplanet.sso.SSOException;
import com.iplanet.sso.SSOToken;
import
com.sun.identity.authentication.spi.AMPostAuthProcessInterface;
import
com.sun.identity.authentication.spi.AuthenticationException;

/**
 * Set a session property on successful authentication.
 * If authentication fails, log a debug message.
 */
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";

    private Logger debug =
LoggerFactory.getLogger(SamplePAP.class);

    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 PingAM? 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.

Configure 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 the AM admin UI, go 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 click **Save**.

Alternatively, you can configure sample plugin for the realm by using the **`ssoadm`** command.

```
$ ssoadm set-svc-attrs \
 --adminid uid=amAdmin,ou=People,dc=openam,dc=forgerock,dc=org \
 --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.
```

Test 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.

Success and failure redirection URLs

AM determines the redirection URL based on authentication success or failure.

You can configure success and failure URLs in a number of places in AM. For each authentication outcome, the user is redirected to the URL with the highest precedence, which is determined by where it is defined in AM.

Success URL precedence

When a user authenticates successfully, AM evaluates the redirection URL according to the following order of precedence:

1. The URL set in the `goto` login URL parameter. For example:

   `http://openam.example.com:8080/openam/XUI/?realm=/alpha&goto=http%3A%2F%2Fwww.example.com#login`

   NOTE

   Unless the URLs are in the same domain as Access Management, any URLs specified in the `goto` parameter must be explicitly listed in the Valid goto URL Resources property of the Validation Service. Otherwise, they're ignored.

2. The URL set in the authentication tree or authentication chain.
   - To specify a URL in an authentication tree, add a Success URL node to the tree and configure the **Success URL** in the node properties.
   - To specify a URL in an authentication chain, in the AM admin UI, go to ***Realm Name*** > **Authentication** > **Chains** > ***chain*** > **Settings**. Enter a URL in the **Successful Login URL** field and save your changes.
3. The URL set in the **Success URL** attribute in the user's profile.

   In the AM admin UI, go to ***Realm Name*** > **Identities** > ***identity***. In the **Success URL** field, enter a URL, and save your changes.

   You can also specify the client type by entering `ClientType|URL` as the property value. If the client type is specified, it takes 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.

   Go to **Configure** > **Authentication** > **Core Attributes** > **Post Authentication Processing**. In the **Default Success Login URL** field, enter a URL, and save your 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 Default Success Login URL in the Top Level realm.

Failure URL precedence

When a user fails to authenticate, AM evaluates the redirection URL according to the following order of precedence:

1. The URL set in the `gotoOnFail` parameter. For example:

   ```
   https://openam.example.com:8443/openam/XUI/?
   realm=/alpha&gotoOnFail=http%3A%2F%2Fwww.example.com#login
   ```

NOTE

Unless the URLs are in the same domain as Access Management, any URLs specified in the `gotoOnFail` parameter must be explicitly listed in the Valid goto URL Resources property of the Validation Service. Otherwise, they're ignored.

1. The URL set in the authentication tree or authentication chain.
   - To specify a URL in an authentication tree, add a Failure URL node to the tree and configure the failure URL in the node properties.
   - To specify a URL in an authentication chain, in the AM admin UI, go to ***Realm Name*** > **Authentication** > **Chains** > ***chain*** > **Settings**. Enter a URL in the **Failed Login URL** field and save your changes.
2. The URL set in the **Failure URL** attribute in the user's profile.

   In the AM admin UI, go to ***Realm Name*** > **Identities** > ***identity***. In the **Failure URL** field, enter a URL, and save your 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.
3. The URL set in the **Default Failure Login URL** attribute in the Top Level Realm.

   In the AM admin UI, go to **Configure** > **Authentication** > **Core Attributes** > **Post Authentication Processing**. In the **Default Failure Login URL** field, enter a URL, and save your 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 Default Failure Login URL in the Top Level realm.

Configure trusted URLs

Redirection 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 an open redirect.

To configure AM to trust other absolute URLs, add them to the validation service. If they are not added, AM will redirect to the user profile or to the administrator console on log in, and to the default logout page in the UI on log out.

▼ Do I need to add my URL to the validation service?

Consider these example URLS for a deployment configured at `https://am.example.com:8443/am`:

URL	Add to the validation service?
`http://am.example.com:8080/am/XUI/#login`	Yes, the scheme and port are different.
`https://am.example.com:443/am/XUI/#login`	Yes, the port is different.
`/am/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.

Add a URL to the validation service

1. In the AM admin UI, go to **Realms** > ***Realm Name*** > **Services**.

   Note that you can add an instance of the validation service on the Top Level Realm, too.

2. If the **Validation Service** is not in the list of services, configure a new service:

   a. Click **Add a Service**.

   b. From the **Choose a service type** drop-down list, select **Validation Service**.

3. In the **Valid goto URL Resources** field, enter one or more valid URL patterns to allow.

   For example, `http://app.example.com:80/*?*`.

   ▼ *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 `https://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/`.

For more information on pattern matching and wildcard rules, refer to Specifying resource patterns with wildcards.

4. Click **Create** to save your settings.

Validate a `goto` URL

To validate a `goto` URL over REST, use the `/json/users?_action=validateGoto` endpoint (or `/json/realms/root/realms/_Realm Name_/users?_action=validateGoto` to specify a sub realm). For example:

```
$ 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/realms/root/realms/alpha/users?_action=validateGoto'
{
    "successURL":"http://www.example.com/"
}
```

If the URL is valid, the endpoint returns the specified URL as the `successURL` response parameter.

A `goto` URL is considered valid if one of the following is true:

- The URL is configured in the validation service
- The URL is relative
- The URL is encoded

Encoded URLs are treated as relative URLs for the purposes of validation. To be treated as an *absolute* URL, the URL must not be encoded.

If the specified URL is invalid, the endpoint returns the default success URL.

Note that a valid session is optional; you can still call the `validateGoto` endpoint with an expired session.

Configure realm authentication properties

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, go to **Configure > Authentication** in the AM admin UI, and click **Core Attributes**.

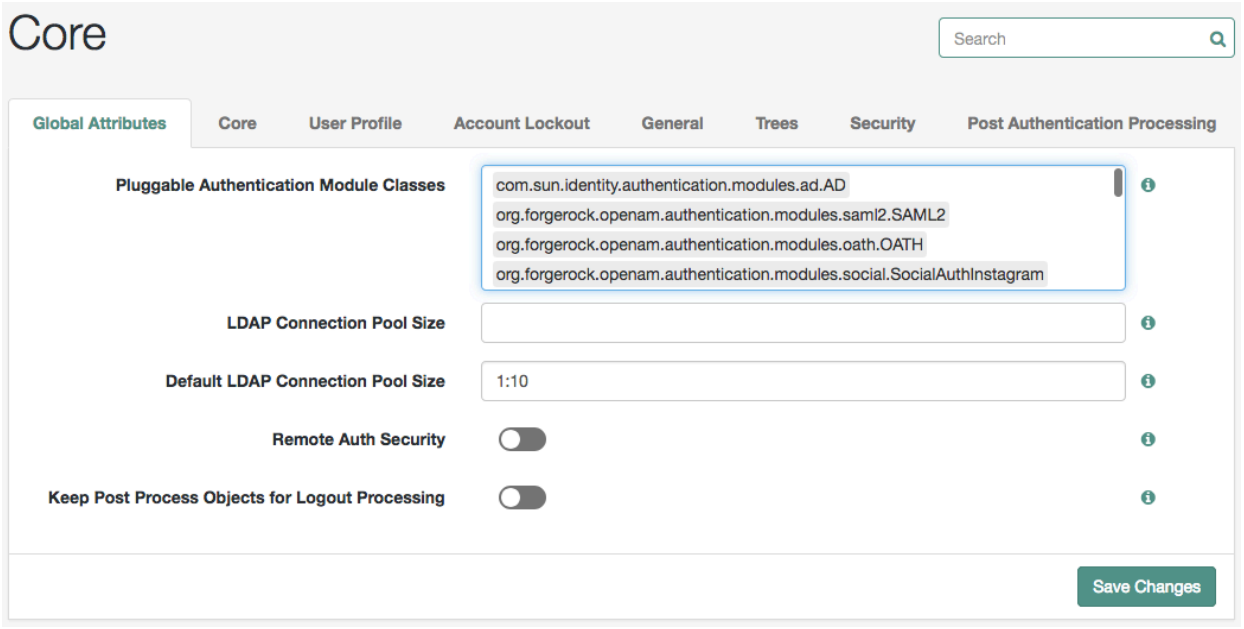


Figure 5. The Core Authentication Attributes Page

To override the global core authentication configuration in a realm, go to **Realms** > ***Realm Name*** > **Authentication** > **Settings** in the AM admin UI. 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.

Authenticate with a browser

When using AM's extended user interface (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/`.

When authenticating using a browser, you can send AM a realm and also different authentication parameters that would help you customize the user's experience.

Specify the realm in the URL

When making a request to the UI, 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. 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 UI 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=alpha#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.

Authentication parameters

AM accepts the following parameters in the query string. Except for the `IDToken` parameters, do not set a parameter more than once in a single query.

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 have a valid session.

  When `ForceAuth=true`, 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. In this configuration, session upgrade is *not* supported for client-side sessions. When a user attempts to reauthenticate with a client-side session, they will see a `ForceAuth fails since session is stateless` dialog.

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 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 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

Authentication chains only. This parameter does not apply to authentication trees.

Set the `resourceURL` to the URL of the resource for resource-based authentication.

Resource-based authentication applies when an authorization policy has an environment condition of type **Authentication by Service** or **Authentication by Module Instance**. When the resource URL matches a policy resource, AM finds the chain or module configured in the policy environment settings and 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 condition **Authentication by Service: DataStore**, the following login URL causes AM to use the DataStore service to authenticate the user:

```
https://openam.example.com:8443/openam/XUI/#login?
resource=true&resourceURL=https://www.example.com:443/index.htm
l&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, authenticate 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 schema provided with AM includes this object class and attribute.

For information about identity repository schema, refer to Prepare identity repositories .

As this parameter determines authentication module selection, do not use it with `authlevel` , `module` , or `service` .

Example UI 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 UI 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 `alpha` realm, requesting that AM display the user interface in German.	`https://openam.example.com:8443/openam/XUI/?realm=/alpha&locale=de#login`
Log in to the `alpha` realm using the `myTree` authentication tree, requesting that AM display the user interface in German.	`https://openam.example.com:8443/openam/XUI/?realm=/alpha&locale=de&service=myTree#login`

Authenticate over REST

AM provides the `/json/authenticate` endpoint for authentication, and the `/json/sessions` endpoint for managing sessions and logging out.

The following table summarizes authentication operations you can perform using REST:

Task	Resources
Authenticate to AM Authenticating to AM means logging in to a specific realm and receiving a session token from AM. Add parameters to the authentication request to provide AM with more information about how you want to authenticate.	Log in to AM over REST
Use the session token AM provides you with a session token after authenticating to a realm. Use this token in subsequent calls to AM. For example, when using REST calls to create, modify, or delete configuration objects.	Session token after authentication
Log out of AM Log out your users by sending a `logout` action to the `/json/sessions` endpoint.	Log out of AM over REST
Invalidate sessions Obtain all the sessions for a given user and invalidate them to ensure they are logged out of AM.	Invalidate all sessions for a given user

Log in to AM over REST

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` .

NOTE

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*.

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: Ch4ng31t" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/realms/alpha/authenticate'
{
    "tokenId": "AQIC5w…NTcy*",
    "successUrl": "/openam/console",
    "realm":"/alpha"
}
```

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.

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 known as the session token. For more information about how applications can use the session token, see Session token after 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":"/alpha"
}
```

TIP

TIP

You can request AM to authenticate a user without providing them a session by using the `noSession` parameter. For more information, see Authenticate endpoint parameters.

UTF-8 user Names

To use UTF-8 user names and passwords in calls to the `/json/authenticate` endpoint, base64-encode the string, and 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: Ch4ng31t" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/realms/alpha/authenticate'
{
    "tokenId": "AQIC5w…NTcy*",
    "successUrl": "/openam/console",
    "realm":"/alpha"
}
```

Authenticate 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: Ch4ng31t" \
--header 'Accept-API-Version: resource=2.0, protocol=1.0' \
'https://openam.example.com:8443/openam/json/realms/root/realms/alpha/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.

Return 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 link: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/realms/alpha/authenticate'
```

```
{
  "authId": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJvdGsiOiJ...", ①
  "template": "", ②
  "stage": "DataStore1", ③
  "callbacks": [
    {
      "type": "NameCallback", ④
      "output": [ ⑤
        {
          "name": "prompt",
          "value": " User Name: "
        }
      ],
      "input": [ ⑥
        {
          "name": "IDToken1",
          "value": ""
        }
      ]
```

```
    },
    {
      "type": "PasswordCallback",
      "output": [
        {
          "name": "prompt",
          "value": " Password: "
        }
      ],
      "input": [
        {
          "name": "IDToken2",
          "value": ""
        }
      ]
    }
  ]
}
```

① The JWT that uniquely identifies the authentication context to AM.

② A template to customize the look of the authentication module, if it exists. Learn more in How do I customize the Login page? in the *Knowledge Base*.

③ The authentication module stage where the authentication journey is at the moment.

④ The type of callback. It must be in Supported callbacks.

⑤ The information AM offers about this callback. Usually, this information would be displayed to the user in the UI.

⑥ 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 `Ch4ng31t` 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":"Ch4ng31t"
               }
            ]
         }
      ]
}' \
'https://openam.example.com:8443/openam/json/realms/root/realms/alpha/authenticate'
{
    "tokenId":"AQIC5wM2…U3MTE4NA..*",
    "successUrl": "/openam/console",
    "realm":"/alpha"
}
```

On complex authentication journeys, AM may send several callbacks sequentially. Each must be completed and returned to AM until authentication is successful.

For more information about the callbacks AM may return, see Supported callbacks.

Session token after authentication

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 server-side sessions are enabled, the `tokenId` object is a reference to the session state stored in the CTS token store.
- If client-side 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-side sessions—2000 bytes or greater—is considerably longer than for server-side sessions—approximately 100 bytes. For more information about session tokens, see Session cookies and session security.

The following is a common scenario when accessing AM by using REST API calls:

1. 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: Ch4ng31t" \
   --header "Accept-API-Version: resource=2.0, protocol=1.0" \
   'https://openam.example.com:8443/openam/json/realms/root/realms/alpha/authenticate'
   {
       "tokenId":"AQIC5wM…TU3OQ*",
       "successUrl":"/openam/console",
       "realm":"/alpha"
   }
   ```

   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.

2. 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 admin UI, by navigating to **Deployment > Servers > *Server Name* > Security > Cookie**, in the **Cookie Name** field of the AM admin UI.

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 Privileges.

3. Call the REST API to log the user out of AM, as described in Authenticate over 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.

Log out of AM over REST

Authenticated users can log out with the token cookie value and an HTTP POST to `/json/sessions/?_action=logout`:

```
$ curl \
--request POST \
--header "iPlanetDirectoryPro: AQICS…NzEz*" \
--header "Accept-API-Version: resource=3.1, protocol=1.0" \
https://openam.example.com:8443/openam/json/realms/root/realms/alpha/sessions/?_action=logout
```

```
{
    "result":"Successfully logged out"
}
```

Invalidate 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 `alpha` realm, the query filter value would be:

```
username eq "demo" and realm eq "/alpha"
```

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 is one active session:

```
$ curl \
--request GET \
--header "iPlanetDirectoryPro: AQICS…NzEz*" \
--header "Accept-API-Version: resource=3.1, protocol=1.0" \
https://openam.example.com:8443/openam/json/realms/root/realms/alpha/sessions?_queryFilter=username%20eq%20%22demo%22%20and%20realm%20eq%20%22%2Falpha%22
{
    "result": [
        {
            "_rev": "652365455",
            "username": "demo",
            "universalId":
"id=demo,ou=user,dc=openam,dc=forgerock,dc=org",
            "realm":"/alpha",
            "sessionHandle":"shandle:cmvShY1….AA.*",
            "latestAccessTime":"2019-10-03T09:36:53.041Z",
            "maxIdleExpirationTime":"2019-10-03T10:06:53Z",
```

```
            "maxSessionExpirationTime":"2019-10-03T11:36:53Z",
        }
    ],
    "resultCount": 1,
    "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 "iPlanetDirectoryPro: AQICS…NzEz*" \
--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/realms/alpha/sessions/?_action=logoutByHandle
{
    "result": {
        "shandle:SJ80.AA….JT.": true,
        "shandle:H4CV.DV….FM.": true
    }
}
```

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. (If you are unfamiliar with HTTP cookies, see HTTP cookies for more information.)

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`.

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 IG documentation.
- CDSSO supports server-side and client-side 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:

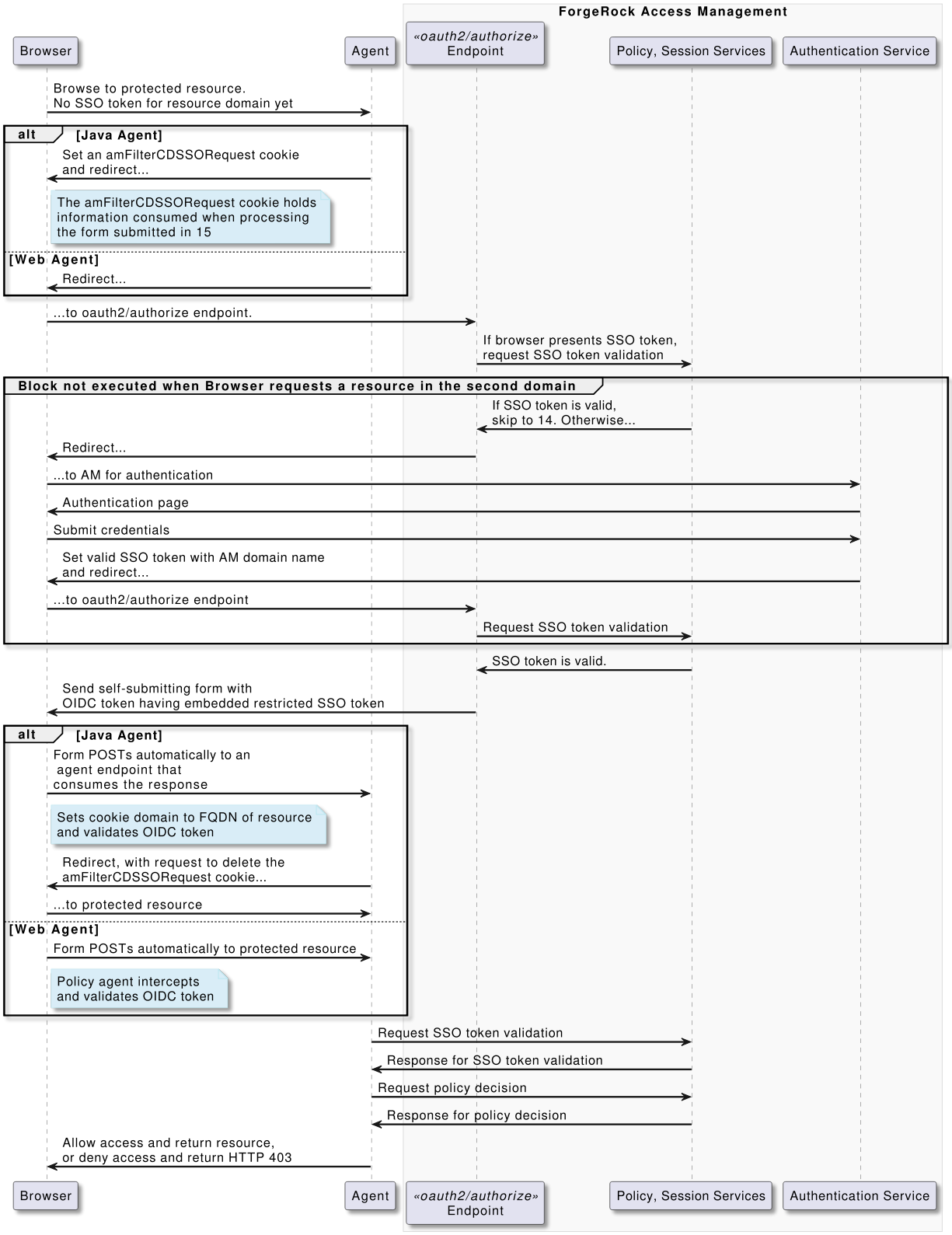


Figure 6. Web and Java agents CDSSO flow

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 end user UI, 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.

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 properties 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.

  For more information, see Secure cookies by default.

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 HttpOnly session cookies.

Expires
: The lifetime of a cookie can be limited, with an `Expires` name configured with a time, based on UTC (GMT).

WARNING

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`.

Implement CDSSO

CDSSO provides SSO capabilities for AM servers and web or Java agents within a single domain or across domains in the same organization.

CDSSO is the only mode of operation for web and Java agents, so no additional configuration is required to make it work.

You must, however, protect the session cookie against hijacking. For more information, see Restrict tokens for CDSSO session cookies.

TIP

IG also supports CDSSO with AM. For more information, see the ForgeRock Identity Gateway Gateway Guide.

Troubleshoot 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 `Cookie Security` or the `CDSSO Secure Enable` properties are configured in the agent profile with a regular HTTP application.

  If you need encrypted communications for an application protected by AM, use the `Cookie Security` or the `CDSSO Secure Enable` properties and make sure the application is accessible over HTTPS.

- 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-side session cookie exceeds the maximum size permitted by the browser.

  As described in Session cookies and session security, 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.

Social authentication

AM supports delegated authentication through third-party identity providers, such as Facebook, and Google. This lets users log in to AM using their social provider credentials.

TIP

This page shows you how to configure social authentication in a standalone AM deployment. To configure social authentication in the ForgeRock Identity Platform, see the ForgeRock Identity Platform Self-Service Guide.

These topics describe the high-level steps to configure social authentication.

Configure social identity providers

AM supports social identity providers that are OAuth 2.0 or OpenID Connect 1.0-compliant. A number of social identity providers are configured by default:

Default social identity provider configurations

Amazon	Apple	Facebook
Google	Instagram	itsme[1]
LinkedIn	Microsoft	Salesforce
Twitter	VK (Vkontakte)	WeChat
WordPress	Yahoo	

[1] To integrate with itsme, you must obtain an Organization Validation (OV) certificate.

You must also configure it in the container where AM runs, or in the reverse proxy offloading SSL.

You can add providers that are not configured by default, as long as these providers have a solution implemented using either OAuth 2.0, or OpenID Connect.

Add identity providers

1. Register a service in the identity provider, and keep the provider's documentation within reach. You will use it during this procedure.

   To register a service in a provider, you must *at least* create a client ID and add the redirect URL to AM.

   ▼ ***Redirect URLs***

   A *redirect URL* is a path in AM to which the identity provider redirects the user on successful authentication. For example, `https://platform.example.com:8443/am`.

   Depending on the social identity provider and on your environment, you might need to make changes to the redirect URL later.

   Configure the same redirect URL in the identity provider service and in the AM client.

IMPORTANT

IMPORTANT

Some providers require that you enable a specific API in their service:

Google
: Enable the `Gmail API` in the Google Cloud Platform.

Apple
: You must have access to the Apple Development Program (Enterprise program is not eligible), and you must enable `Sign In With Apple` in the Apple Developer site.

Twitter
: You must have an Elevated Developer Twitter Account to obtain a token, and you must set up an application at https://developer.twitter.com/en/portal/dashboard.

2. In the AM admin UI, go to **Realms** > ***Realm Name*** > **Services**.
3. Check if the `Social Identity Provider Service` appears in the list of services configured for the realm.

   If it does not, click **Add a Service**, select `Social Identity Provider Service` from the drop-down list, and click **Create**.
4. Ensure that the **Enabled** switch is on.
5. Go to the **Secondary Configurations** tab.

   AM includes scripts and configurations for several common identity providers.
6. In the **Add a Secondary Configuration** drop-down list, select the required identity provider.

   If you do not see the required provider, select one of the following to add a custom identity provider client:

   - **Client Configuration for providers that implement the OAuth2 specification**
   - **Client Configuration for providers that implement the OpenID Connect specification**
7. Provide the details of the service you registered with the social provider:
   a. In the **Client ID** field, enter the client ID of the service. This field might have a different name, depending on the social identity provider. For example, for Twitter, this field is the **API key**.
   b. In the **Client Secret** field, enter the client secret of the service. This field might have a different name, depending on the social identity provider. For example, for Twitter, this field is the **API secret**.
   c. Enter the same **Redirect URL** that you set in the social provider service.

▼ *Redirect URLs*

A *redirect URL* is a path in AM to which the identity provider redirects the user on successful authentication. For example, `https://platform.example.com:8443/am`.

Depending on the social identity provider and on your environment, you might need to make changes to the redirect URL later.

Configure the same redirect URL in the identity provider service and in the AM client.

Don't worry if you are missing some of the details; you'll be able to edit the configuration later, after saving the client profile for the first time.

Click **Create** to access all the configuration fields for the client.

8. Provide the client's advanced configuration details, and edit any required configuration details if needed.

▼ *Where do I find the required identity provider information?*

- Refer to the provider's documentation.

  Providers must specify their integration needs in their documentation, as well as their API endpoints.

  For example, providers usually have different scopes that you can configure depending on your service's needs.

  Financial-grade providers usually also require additional security-related configuration, such as `acr` values, PKCE-related settings, and more.

  Keep their documentation close while configuring the client profile.

- Visit the provider's `.well-known` endpoint.

  OAuth 2.0/OpenID Connect-compliant providers will display much of the information you need to configure the identity provider client in their `.well-known` endpoint. For example, the endpoint should expose their endpoint URLs, and the signing and encryption algorithms they support.

AM provides a default configuration for the supported social identity providers based on each provider's requirements. Providers sometimes change their requirements over time. **Make sure the settings for the provider have not changed.**

The important preconfigured fields are:

- The provider's URLs. For example, **Authentication Endpoint URL** , **Access Token Endpoint URL** , and **User Profile Service URL** .
- The **OAuth Scopes** field.

- The configuration in the **UI Config Properties** section.
- The script selected in the **Transform Script** drop-down list.

  This script is responsible for mapping attributes provided by the identity providers to a profile format compatible with AM.

  For details, refer to Transform Script.

NOTE

Some features require choosing algorithms from those supported by the provider, as well as creating secrets. Consider the following points before configuring the client:

- Several capabilities in the identity provider client share the same secret IDs. For example, signing request objects and signing client authentication JWTs.
- Every identity provider client in a realm shares the same secrets.

Therefore, ensure that you configure features requiring secrets in a way that they are compatible across clients in the same realm.

For more information, see the page about the /oauth2/connect/rp/jwk_uri endpoint.

For tips on how to configure the client, see Client configuration reference.

9. Save your changes.

You are now ready to Configure basic social registration trees.

TIP

To let AM contact Internet services through a proxy, see Configuring AM for Outbound Communication.

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 Advanced Properties.

Configure basic social registration trees

There are two nodes associated with Identity Providers:

Select Identity Provider node

The Select Identity Provider node prompts the user to select a social identity provider to register or log in with, or (optionally) continue on with a local registration or login flow. When a provider is selected, the flow continues on to the Social Provider Handler node.

Social Provider Handler node

The Social Provider Handler node is used in combination with the Select Identity Provider node. It communicates with the selected provider and collects the information provided after the user has authorized the service. It then takes that information and runs a transformation script to prepare it.

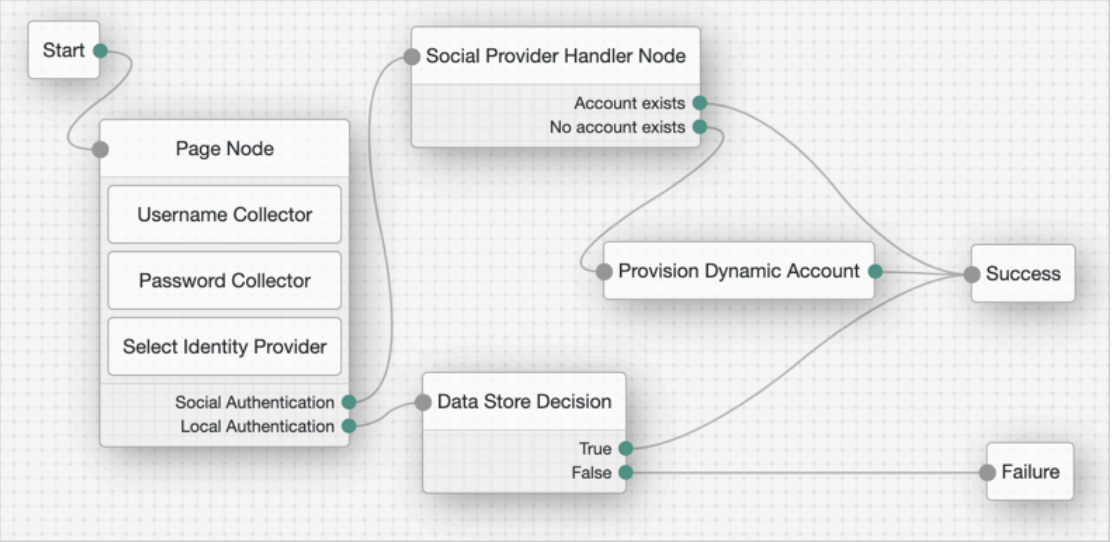
AM includes a transformation script called **Normalized Profile to Identity**, which this node uses to transform the identity object gathered from the identity provider into a user profile in AM's identity store.

The node then queries IDM to see if the user already exists. If the user exists, they are logged in. If the user does not exist, the user will need to be created.

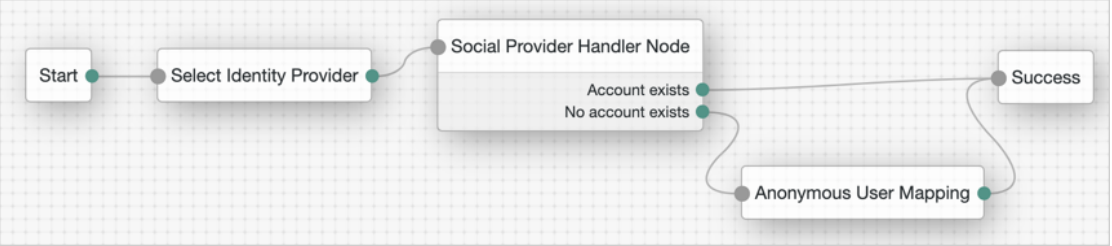
The node then queries the identity store available for the realm to see if the user already exists. If the user exists, they are logged in. If the user does not exist, the user will need to be created.

Set up a basic social registration tree

1. In the AM admin UI, go to **Realms** > ***Realm Name*** > **Authentication** > **Trees**, and create a new tree.
2. Decide whether users can log in with their AM credentials, and add the relevant nodes to the tree:
   a. Social authentication trees allowing local authentication might look like the following:



   b. Social authentication trees enforcing social authentication login might look like the following:



   To configure either option, use the **Include local authentication** switch in the Select Identity Provider node.

To support both local and social authentication in the same page, you must use the Page node as shown in the example.

3. Configure the Social Provider Handler node:
   - In the **Transformation Script** field, select `Normalized Profile to Identity`. This script will transform the normalized identity provider's profile object into the appropriate user profile attributes of the realm's identity store.

     If you are not using DS as the identity store, or if you added customized fields to it, you may need to modify the script.

     Find the script in Realms > *Realm Name* > Scripts.
   - In **Client Type**, select `BROWSER` when using the AM UI, or the ForgeRock SDK for JavaScript.

Client configuration reference

Enabled

Specifies whether the provider is enabled.

Required: Yes.

Auth ID Key

Specifies the attribute the social identity provider uses to identify an authenticated individual. For example, `id`, `sub`, and `user_id`.

Required: Yes.

Client ID

Specifies the `client_id` parameter as described in section 2.2 of *The OAuth 2.0 Authorization Framework* specification.

Required: Yes.

Client Secret

Specifies the `client_secret` parameter as described in section 2.3 of *The OAuth 2.0 Authorization Framework* specification.

Required: No.

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*. For example, `https://accounts.google.com/oauth2/v2/auth`.

Required: Yes.

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* specification. For example, `https://www.googleapis.com/oauth2/v4/token`.

Required: Yes.

User Profile Service URL

Specifies the user profile URL that returns profile information. For example, `https://www.googleapis.com/oauth2/v3/userinfo`.

This URL should return JSON objects in its response.

Required: No.

Token Introspection Endpoint URL

Specifies the URL to the endpoint handling access token validation, as described in the OAuth 2.0 Token Introspection specification. For example, `https://oauth2.googleapis.com/tokeninfo`.

Required: No.

Redirect URL

Specifies the URL the identity provider will redirect the user to after authenticating, as described in Section 3.1.2 of *The OAuth 2.0 Authorization Framework* specification.

This URL is usually a page or path in AM; for example, `https://platform.example.com:8443/am`, and it is also registered in the identity provider's service.

You can also use a custom URI scheme as the redirect, if you are using an app built with the ForgeRock SDKs for Android or iOS. For example, `com.example.sdkapp:redirect_uri_path` or `frauth://com.forgerock.ios.sdkapp`.

TIP

When using the `FORM_POST` **Response Mode**, you must specify the `form_post` endpoint in the redirection URL. See **Response Mode** for more information.

Required: Yes.

Redirect after form post URL

Specifies the URL of a custom login page or application. AM will send processed form post data related to social login authentication to that URL as the value of the `form_post_entry` query parameter.

To continue the authentication journey, the custom login page is responsible for making a call to the AM `/json/authenticate` endpoint with the authentication ID (`authID`) and the processed form data (`form_post_entry`).

Configure this property when the following is true:

- The `FORM_POST` **Response Mode** is configured.
- Your users log in to AM using custom login pages, such as apps using the ForgeRock SDKs, instead of the AM UI.

  Required: No.

Scope Delimiter

Specifies the delimiter used to separate scope values. For example, a blank space ( ), or a comma character (`,`).

Most providers use a blank space.

Required: Yes.

OAuth Scopes

Specifies the list of scopes to request from the provider.

The scopes that the provider returns depends on the permissions that the resource owner, such as the end user, grants to the client application.

For example, Google exposes its supported scopes in their OAuth 2.0 Scopes for Google APIs documentation.

Required: Yes.

Client Authentication Method

Specifies how the client should authenticate to the provider. Possible values are:

- `CLIENT_SECRET_POST` . The client sends the client ID and the secret in the `client_ID` and the `client_secret` parameters in the body of the request.
- `CLIENT_SECRET_BASIC` . The client sends the client ID and the secret in a basic authorization header with the base64-encoded value of *client-id:client-secret*.
- `PRIVATE_KEY_JWT` . The client sends its credentials to the provider in a signed JWT as specified in the JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants.
- `ENCRYPTED_PRIVATE_KEY_JWT` . The client sends its credentials to the provider in a signed, then encrypted JWT as specified in the JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants.
- `TLS_CLIENT_AUTH` . The client presents a X.509 certificate that uses public key infrastructure (PKI), as specified in the OAuth 2.0 Mutual TLS (mTLS) Client Authentication and Certificate Bound Access Tokens.
- `SELF_SIGNED_TLS_CLIENT_AUTH` . The client presents a X.509 self-signed certificate, as specified in the OAuth 2.0 Mutual TLS (mTLS) Client Authentication and Certificate Bound Access Tokens.

Some of the authentication methods require additional configuration:

▼ *How do I configure JWT authentication with signed JWTs?*

a. Obtain a list of supported signing algorithms from the provider's `.well-known` endpoint, and decide which one you will use.

b. In the **Private Key JWT Signing Algorithm** field, enter the signing algorithm that AM will use to sign the JWT. For example, `RSA256`.

   This field may already be configured if the client is sending request objects.

c. Create a signing secret, and map it to the `am.services.oauth2.oidc.rp.jwt.authenticity.signing` secret ID in an AM secret store.

   The secret ID may already have secrets mapped to it if the client is sending signed request objects to the provider, or if another client in the realm is already using it.

   For more information, see Configuring Secret Stores, and /oauth2/connect/rp/jwk_uri.

d. Provide a JWK with the public key to the identity provider. Refer to their documentation for more information.

   For example, you could copy the contents of the public JWK in a field in the provider's service configuration, or you could configure the realm's `/oauth2/connect/rp/jwk_uri` endpoint, which exposes the client's public keys.

   Configure the realm's `/oauth2/connect/rp/jwk_uri` endpoint in the provider, which exposes the client's public keys. Refer to the provider's documentation for more information.

e. Change the value in the **Private Key JWT Expiration Time (seconds)** field, if needed. It has a sensible value preconfigured, but you may need to tune it for your provider.

▼ *How do I configure JWT authentication with signed and encrypted JWTs?*

a. Follow the steps in How do I configure JWT authentication with signed JWTs? to configure AM to sign authentication JWTs.

   Now you are ready to configure AM to encrypted authentication JWTs.

b. Obtain a list of supported encryption algorithms and methods from the provider's `.well-known` endpoint, and decide which one you will use.

c. In the **JWT Encryption Algorithm** field, select the encryption algorithm.

If the required encryption algorithm does not appear in the drop-down, check the reference entry for the **JWT Encryption Algorithm** field for information on how to add it.

This field may already be configured if the client is encrypting request objects.

d. In the **JWT Encryption Method** field, select the encryption method.

This field may already be configured if the client is encrypting request objects.

e. In the **JWKS URI Endpoint** field, configure the URI containing the provider's public JWK set.

Obtain the URI from the provider's `.well-known` endpoint, or their documentation.

AM will use the JWK URI to fetch the provider's public encryption key.

f. Perform one of the following steps depending on the encryption method you configured:

i. If you chose Direct AES Encryption method, select `NONE` in the **JWT Signing Algorithm** field. Signing is redundant with this encryption method.

ii. If you chose an encryption method different from the Direct AES Encryption method, configure signing. For more information, see How do I configure JWT authentication with signed JWTs?.

▼ *How do I configure mTLS authentication with PKI?*

a. Obtain a certificate for AM to use as a client. This certificate must be signed by a certificate authority (CA).

You can use the same certificate for different social identity provider client configurations, and you can only have one mTLS certificate by realm (either PKI-related, or self-signed).

b. Make the certificate available to AM configuring it in an AM secret store, and map its alias to the `am.services.oauth2.mtls.client.authentication` secret ID.

For example, you can create a `PKCS12` keystore secret store.

For more information, see Configure secret stores.

Even though the identity provider should trust the CA certificate automatically, the client certificate will appear in the /oauth2/connect/rp/jwk_uri endpoint.

▼ *How do I configure mTLS authentication with self-signed certificates?*

a. Obtain a self-signed certificate that AM will use as a client.

   You can use the same certificate for different social identity provider client configurations, and you can only have one mTLS certificate by realm (either PKI-related, or self-signed).

b. Make the certificate available to AM configuring it in an AM secret store, and map its alias to the `am.services.oauth2.mtls.client.authentication` secret ID.

   For example, you can create a `PKCS12` keystore secret store.

   For more information, see Configuring Secret Stores.

   To trust the self-signed certificate, the social identity provider must be able to access its public key and certificate. Social identity providers may have different ways of accessing public keys; for example, you may be able to configure the public JWK directly in the provider, or you may be able to provide AM's /oauth2/connect/rp/jwk_uri endpoint, which exposes it.

   Refer to your social identity provider documentation for more information.

Required: Yes.

PKCE Method

Specifies the PKCE transformation method AM uses when making requests to the provider's authorization endpoint, as specified in Section 4.2 of the *Proof Key for Code Exchange by OAuth Public Clients* specification.

Select `NONE` to disable PKCE transformations.

Required: No.

Request Parameter JWT Option

(OpenID Connect providers only) Specifies whether AM should provide a request object JWT to the provider. Possible values are:

- `NONE` . AM does not send a request object to the provider.
- `REFERENCE` . The request object JWT is stored in AM's CTS token store, and AM exposes a unique identifier for it using the `oauth2/request_uri` endpoint for the realm. The URL to the endpoint and the JWT's unique identifier are passed to the provider in the `request_uri` parameter of the request.

  Ensure that the provider can reach the endpoint.

  An example of the URL is `https://platform.example.com:8443/am/realms/root/realms/myRealm/oauth2/request_uri/requestobjectID`

NOTE

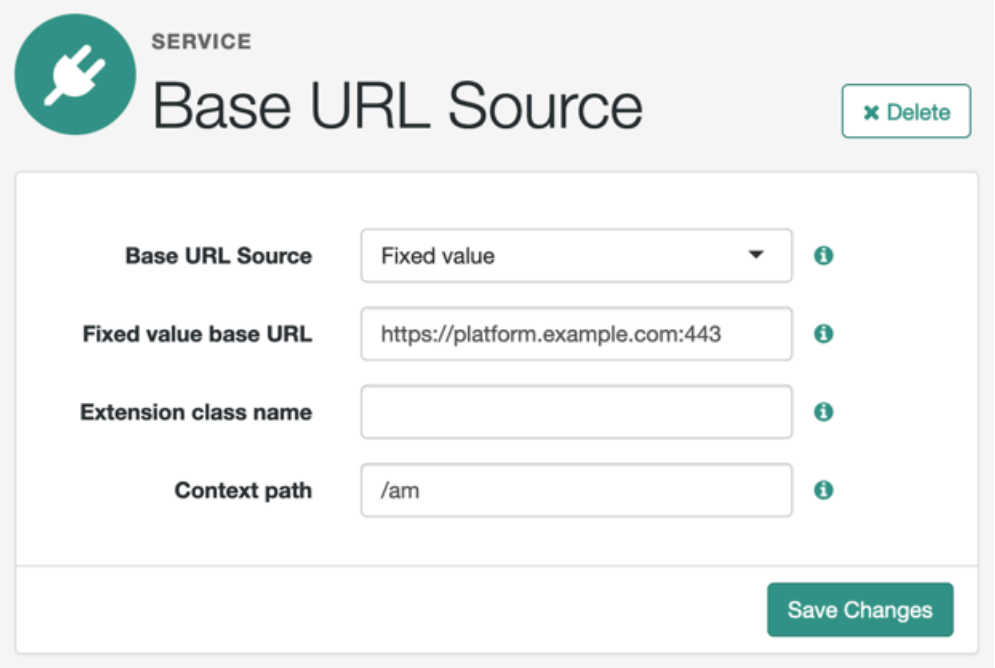
NOTE

When integrating with itsme, ensure that the base URL of AM contains the `443` port. For example, `https://platform.example.com:443/am`.

To do this, configure the reverse proxy or load balancer to expose the port, or the Base URL Source Service:

i. In the AM admin UI, go to **Realms** > ***Realm Name*** > **Services**.

ii. Add a **Base URL Source** service if one is not already configured, or select it to change its properties:



- `VALUE`. AM appends the JWT as the value of the `request` parameter of the request.

▼ *How do I configure the client to send signed request objects?*

a. In the **Request Parameter JWT Option** field, select either `VALUE` or `REFERENCE`.

Refer to your identity provider's documentation for more information.

b. Obtain a list of supported signing algorithms from the provider's `.well-known` endpoint, and decide which one you will use.

c. In the **JWT Signing Algorithm** field, select the signing algorithm that AM will use to sign the request object. For example, `RS256`.

This field may already be configured if the client is using JWT client authentication.

d. Create a signing secret that uses the algorithm you selected previously, and map it to the`am.services.oauth2.oidc.rp.jwt.authenticity.signing` secret ID in an AM secret store.

The secret ID may already have secrets mapped to it if the client is using JWT client authentication, or if another client in the realm is already using it.

For details, refer to Secret stores and /oauth2/connect/rp/jwk_uri.

e. Provide a JWK with the public key to the identity provider. Refer to their documentation for more information.

For example, you could copy the contents of the public JWK in a field in the provider's service configuration, or you could configure the realm's `/oauth2/connect/rp/jwk_uri` endpoint, which exposes the client's public keys.

Configure the realm's `/oauth2/connect/rp/jwk_uri` endpoint in the provider, which exposes the client's public keys. Refer to the provider's documentation for more information.

▼ *How do I configure the client to send signed and encrypted request objects?*

a. Follow the steps in How do I configure the client to send signed request objects? to configure AM to send signed request objects.

Now you are ready to configure AM to send encrypted request objects.

b. Enable **Encrypt Request Parameter JWT**.

c. Obtain a list of supported encryption algorithms and methods from the provider's `.well-known` endpoint, and decide which one you will use.

d. In the **JWT Encryption Algorithm** field, select the encryption algorithm.

If the required encryption algorithm does not appear in the drop-down, check the reference entry for the **JWT Encryption Algorithm** field for information on how to add it.

This field may already be configured if the client is encrypting authentication JTWs.

e. In the **JWT Encryption Method** field, select the encryption method.

This field may already be configured if the client is encrypting authentication JWTs.

f. In the **JWKS URI Endpoint** field, configure the URI containing the provider's public JWK set.

Obtain the URI from the provider's `.well-known` endpoint.

AM will use the JWK URI to fetch the provider's public encryption key.

g. Perform one of the following steps depending on the encryption method you configured:

i. If you chose Direct AES Encryption method, select `NONE` in the **JWT Signing Algorithm** field. Signing is redundant with this encryption method.

ii. If you chose an encryption method different from the Direct AES Encryption method, configure signing. For more information, see How do I configure the client to send signed request objects?.

Encrypt Request Parameter JWT

Specifies whether the request parameter must be encrypted when **Request Parameter JWT Option** is set to `REFERENCE` or `VALUE`.

ACR Values

(OpenID Connect providers only) Specifies a space-separated list, in order of preference, of the client's `acr` values.

Required: No.

Well Known Endpoint

(OpenID Connect providers only) Specifies the URL for retrieving information about the provider, such as endpoints, and public keys. For example, `https://accounts.google.com/.well-known/openid-configuration`.

Required: Yes.

Request Object Audience

(OpenID Connect providers only) Specifies the intended audience (`aud`) of the request object when the **Request Parameter JWT Option** field is set to `VALUE` or `REFERENCE`.

When not configured, the value of the **Issuer** field will be used as the audience of the request object.

OP Encrypts ID Tokens

(OpenID Connect providers only) Specifies whether the provider encrypts ID Tokens.

▼ How do I configure the AM to receive encrypted tokens?

a. Obtain a list of supported ID token encryption algorithms from the provider's `.well-known` endpoint, and decide which one the client will use.

b. Create a suitable secret for the algorithm that you chose, and map it to the `am.services.oauth2.oidc.rp.idtoken.encryption` secret ID in an AM secret store.

The secret ID may already have secrets mapped if another client in the realm is already using it.

For more information, see Configuring Secret Stores, and /oauth2/connect/rp/jwk_uri.

c. Provide a JWK with the public key to the identity provider. Refer to their documentation for more information.

   For example, you could copy the contents of the public JWK in a field in the provider's service configuration, or you could configure the realm's `/oauth2/connect/rp/jwk_uri` endpoint, which exposes the client's public keys.

   Configure the realm's `/oauth2/connect/rp/jwk_uri` endpoint in the provider, which exposes the client's public keys. Refer to the provider's documentation for more information.

Required: No.

Issuer

(OpenID Connect providers only) Specifies the issuer of ID Tokens. Must exactly match the value returned in the ID token.

Obtain the `issuer` value from the provider's `.well-known` endpoint.

Required: Yes.

Enable Native Nonce

(OpenID Connect providers only) When enabled, the provider native SDK must include a `nonce` claim in the ID token. The value of the claim must be the value of the `nonce` claim sent in the Authentication Request.

Required: No.

User Info Response Format

(OpenID Connect providers only) Specifies the format in which the provider's `userinfo` endpoint returns data.

Some of the options require additional configuration:

▼ How do I configure the client to receive signed userinfo JWTs?

a. In the **JWKS URI Endpoint** field, configure the URL containing the provider's public JWK set. Obtain it from the provider's `.well-known` endpoint, or their documentation.

   AM will use this URL to fetch the provider's public signing key.

▼ How do I configure the client to receive signed, then encrypted userinfo JWTs?

a. Follow the steps in How do I configure the client to receive signed userinfo JWTs? to configure AM to receive signed JWTs.

   Now you are ready to configure AM to receive encrypted JWTs.

b. Obtain a list of supported ID token encryption algorithms from the provider's `.well-known` endpoint, and decide which one the client will use.

c. Create a suitable secret for the algorithm that you chose, and map it to the `am.services.oauth2.oidc.rp.idtoken.encryption` secret ID in an AM secret store.

   The secret ID may already have secrets mapped if another client in the realm is already using it, or if the provider encrypts ID tokens.

   For more information, see Configuring Secret Stores, and /oauth2/connect/rp/jwk_uri.

d. Provide a JWK with the public key to the identity provider. Refer to the identity provider's documentation for more information.

   For example, you could copy the contents of the public JWK in a field in the provider's service configuration, or you could configure the realm's `/oauth2/connect/rp/jwk_uri` endpoint, which exposes the client's public keys.

   Configure the realm's `/oauth2/connect/rp/jwk_uri` endpoint in the provider, which exposes the client's public keys. Refer to the provider's documentation for more information.

Possible values are:

- `JSON`. The provider's `userinfo` endpoint returns a JSON object.
- `SIGNED_JWT`. The provider's `userinfo` endpoint returns a signed JWT.
- `SIGNED_THEN_ENCRYPTED_JWT`. The provider's `userinfo` endpoint returns a signed, then encrypted JWT.

JWKS URI Endpoint

Specifies the URI that contains the public keys of the identity provider. AM will use these keys to verify signatures, or to encrypt objects.

Configure this field when:

- **Client Authentication Method** is set to `ENCRYPTED_PRIVATE_KEY_JWT`.
- **Encrypt Request Parameter JWT** is enabled.
- **User Info Response Format** is set to `SIGNED_JWT` or `SIGNED_THEN_ENCRYPTED_JWT`.

Required: No.

Claims

Any claims on the request object, in JSON format. These claims must conform to the claims request parameter, as defined in the *OpenID Connect specification*.

JWT Signing Algorithm

Specifies the signing algorithm supported by the provider that AM use to sign the following:

- Client authentication JWTs when **Client Authentication Method** is set to `PRIVATE_KEY_JWT`.
- (OpenID Connect providers only) Request JWTs when **Request Parameter JWT Option** is set to `VALUE` or `REFERENCE`.

Obtain a list of the supported algorithms from the provider's `.well-known` endpoint. Select `NONE` if the client will encrypt the JWT with the Direct AES Encryption method, because the signature will be redundant.
Required: No.

JWT Encryption Algorithm

Specifies the encryption algorithm supported by the provider that AM should use to encrypt client authentication JWTs when **Client Authentication Method** is set to `PRIVATE_KEY_JWT`, and (OpenID Connect providers only) request JWTs when **Request Parameter JWT Option** is set to `VALUE` or `REFERENCE`.

If set to `NONE`, AM will not encrypt the JWTs. Obtain a list of the supported algorithms from the provider's `.well-known` endpoint. Configure the algorithms exposed in this field using the AM advanced server property, openam.private.key.jwt.encryption.algorithm.whitelist.

▼ How do I configure advanced server properties?

- To configure advanced server properties for all instances in the AM environment, go to **Configure > Server Defaults > Advanced** in the AM admin UI.
- To configure advanced server properties for a specific instance, go to **Deployment > Servers > *Server Name* > Advanced**.

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.

Required: No.

JWT Encryption Method

Specifies the encryption algorithm supported by the provider that AM should use to encrypt the following:

- Client authentication JWTs when **Client Authentication Method** is set to `PRIVATE_KEY_JWT`.
- (OpenID Connect providers only) Request JWTs when **Request Parameter JWT Option** is set to `VALUE` or `REFERENCE`.

Use in conjunction with `JWT Encryption Algorithm`. Obtain a list of the supported methods from the provider's `.well-known` endpoint.
Required: No.

Private Key JWT Expiration Time (seconds)

Specifies the amount of time, in seconds, that AM will cache the client authentication JWT before creating a new one.

Caching the JWT avoids creating a new one for every client authentication. However, it may also become invalid if the provider changes it configuration.

Required: No.

Response Mode

(OpenID Connect providers only) Specify the way the provider will return ID tokens to AM. Possible values are:

- `DEFAULT`. The provider returns the ID token as query parameters, as explained in the OpenID Connect Core 1.0 incorporating errata set 1 specification.

  Most preconfigured providers use the `DEFAULT` response mode.

- `FORM_POST`. The provider returns the ID token by submitting an HTML form using the HTTP POST method, as explained in the OAuth 2.0 Form Post Response Mode specification.

  When using this response mode, add the `/oauth2/client/form_post/`*`ClientConfigName`* URI to the **Redirect URL**, where *ClientConfigName* is the name of the social identity provider client that you are configuring. For example, `https://platform.example.com:8443/am/oauth2/client/form_post/myAppleClient`.

  By default, the `form_post` endpoint processes the post data, encrypts it, and redirects with it back to the authentication tree to resume authentication.

  However, environments using custom login pages need to configure the **Redirect after form post URL** property to redirect back to the custom login pages.

  IMPORTANT

  The `/oauth2/client/form_post` does not require authentication. Protect it from denial of service (DoS) attacks by limiting the rate at which it can take connections in your load balancer or proxy.

  If you configured AM with AES Key Wrap encryption, you must configure the `org.forgerock.openam.encryption.useextractandexpand` property.

  For more information, see Preparing AES Key Wrap Encryption.

Required: Yes.

Request Native App for UserInfo

(Apple SSO) When enabled, this flag indicates that the native app can send the user's `userinfo` in JSON format.

Apple returns the `userinfo` only *once*, when the user first consents to send their details, and not on subsequent authentication attempts. In addition, the user has the option *not* to consent to Apple sending their `userinfo`.

If you are progressively profiling the `userinfo` with data from other social providers (usually, using a Patch Object node), there is a risk of overwriting the user's details with blank values when the user authenticates through Apple SSO.

To mitigate this risk, you can add a Scripted Decision node to your authentication journey that assesses whether the `userinfo` is provided and patches the object accordingly.

▼ How do I use a Scripted Decision node to patch an object, based on the returned `userinfo`?

The normalized-profile-to-managed-user.groovy script and normalized-profile-to-managed-user.js scripts set a boolean flag (`nameEmptyOrNull`) that indicates whether the user's `firstName` and `lastName` have been returned.

Create a custom script that uses this flag, then add a Scripted Decision node that calls your script. An example of a custom script that achieves this functionality follows:

JavaScript | Groovy

```
if (sharedState.get('nameEmptyOrNull')) {
  outcome = 'true'
} else {
outcome = 'false'
}
```

The outcome of the Scripted Decision node will be either to patch the `userinfo` object or not to patch the `userinfo` object.

If you need to progressively profile the user information on every authentication, *regardless of whether the user's first name and last name are returned by the OIDC provider*, you can use another Scripted Decision node that does the following: * If the user details are not present, route the `userinfo` patch through a Patch Object node, configured to ignore the `firstName` and `lastName` . (In the **Ignored Fields** list, add `givenName` to ignore the `firstName` and `sn` to ignore the

`lastName` .) * If the user details are present, route the `userinfo` patch through a Patch Object node that patches the full object.

For information about using Scripted Decision nodes in authentication journeys, see Scripted Decision node.

Learn more in First name and last name get blanked out when signing in to Advanced Identity Cloud or PingAM using Apple social sign-on in the *Knowledge Base*.

Required: No.

UI Config Properties

Specifies a map of properties defined and consumed in the UI. The map affects how the identity provider's logo will show on the login page.

▼ AM common end user UI properties

- `buttonImage` : A relative path to an image in the End User UI.
- `buttonCustomStyle` : Any custom CSS you wish to apply to the button outside of normal End User UI styling.
- `buttonClass` : Adds the specified class to the identity provider button, for any additional styling you want to apply.
- `buttonCustomStyleHover` : Adds custom styling when the cursor is hovering over the button.
- `buttonDisplayName` : The name of the identity provider, which will be included either on the button or in the button's `alt` attribute, depending on styling.
- `iconFontColor` : Specifies the color of the icon. You can use methods supported in CSS (such as `white` , or `#ffffff`).
- `iconClass` : Adds the specified class to the identity provider icon, for any additional styling you want to apply.
- `iconBackground` : The color for the background of the icon. You can use methods supported in CSS (such as `white` , or `#ffffff`).

Required: Yes.

Transform Script

A script to convert the provider's raw profile object into a normalized object.

Each social identity provider returns different user profile information using their own attribute names.

For example, Google's OIDC `/userinfo` endpoint returns claims, which AM stores in a `rawProfile` object. The following `google-profile-normalization.groovy` script maps the attributes of this object to AM profile attributes:

```
import static org.forgerock.json.JsonValue.field
import static org.forgerock.json.JsonValue.json
import static org.forgerock.json.JsonValue.object

return json(object(
  field("id", rawProfile.sub),
  field("displayName", rawProfile.name),
  field("givenName", rawProfile.given_name),
  field("familyName", rawProfile.family_name),
  field("photoUrl", rawProfile.picture),
  field("email", rawProfile.email),
  field("username", rawProfile.email),
  field("locale", rawProfile.locale)))
```

The script returns a JSON object containing normalized attributes in the following format:

```
("<platformAttributeName>", rawProfile.<providerAttributeName>)
```

For example, `id` is the platform attribute name, while `rawProfile.sub` is the field received from the provider.

Even if field names are the same, such as `email` and `rawProfile.email`, they must still be mapped for them to be included in the returned JSON object.

AM provides default scripts for other preconfigured identity providers. To view the bindings and expected return values, find the *`<identity provider>`*`-profile-normalization.*` scripts in Sample scripts.

To write your own script in Groovy or Javascript for an identity provider, go to **Realms** > ***Realm Name*** > **Scripts**, and use the provided scripts as a reference.

When a user authenticates, the social authentication journey calls another transformation script set in the Social Provider Handler node to convert the attributes again; this time into an identity object that AM can process.

IMPORTANT

IMPORTANT

Social authentication nodes expect every attribute to have a value. In other words, the attributes returned by the identity provider cannot be empty or `null`, or the journey will end with an error.

For example, if a user tries to log in using Google as the identity provider, but they did not configure a surname in their account, Google returns `null` as the value of the `familyName` for the identity, and social authentication fails.

Ensure all users have their social profiles configured correctly, or modify the transformation scripts so that they don't collect null or empty attributes.

Required: Yes

Suspended authentication

Suspended authentication lets you save a user's progress through an authentication tree, and later resume from the same point.

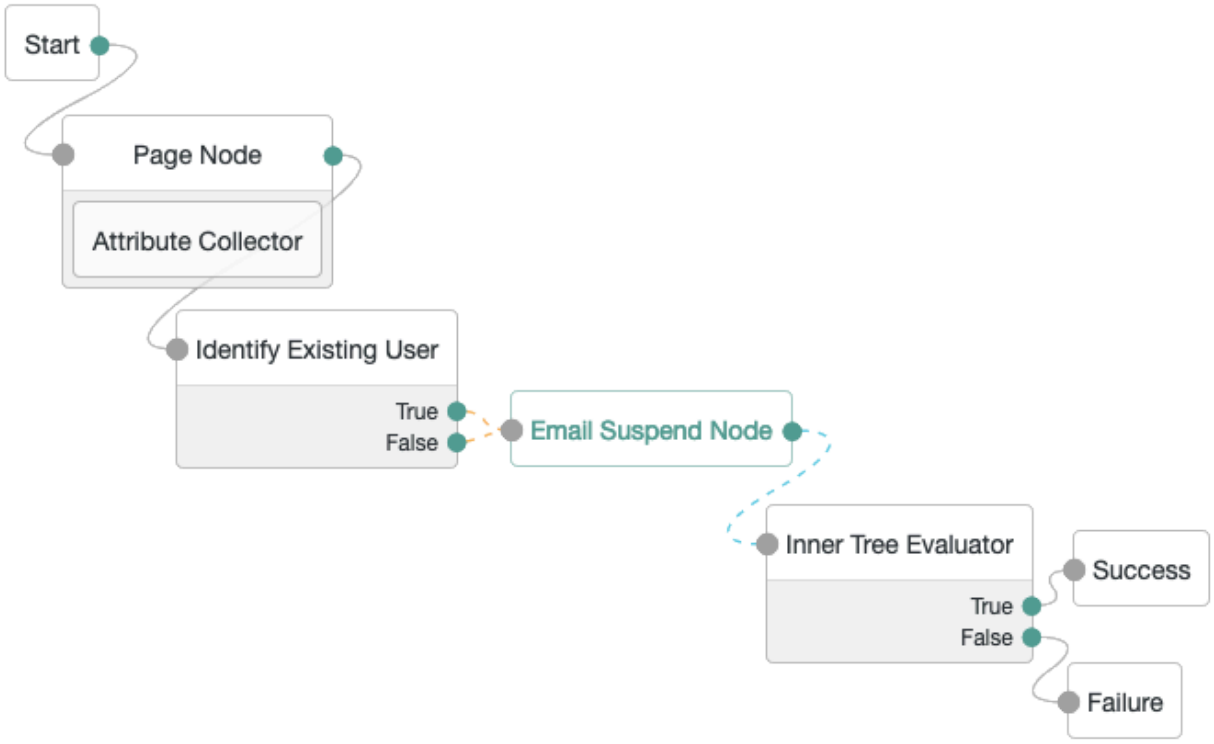
Any input provided during authentication is saved when the authentication tree is suspended, and restored when the authentication tree is resumed. This lets the authentication tree continue after closing the browser, using a different browser, or even on a different device.

When suspending an authentication tree, you provide the user with a URL they must visit to resume their authentication. That URL contains a unique identifier for retrieving the saved progress, and can only be used once. These URLs are sometimes referred to as *magic links*.

The Email Suspend node supports suspended authentication.

Typical use cases include passwordless authentication, and email verification during progressive profile completion.

The following example lets a user authenticate if they have forgotten their username:



After obtaining the user's email address in the Attribute Collector node, the example tree attempts to identify the user. Then, the tree attempts to email the user, and suspends itself.

Note that both the *True* and *False* outcomes are mapped into the Email Suspend node to reduce potential data leakage. If the username is found, it is included in the email sent to the user, along with the link to use to resume the authentication tree.

When the user follows the link, the authentication tree resumes at the Inner Tree Evaluator node, which lets the user authenticate with their recovered username and credentials.

Configure suspended authentication

You can configure the length of time an authentication session can last for in AM, so that resources can be freed up from incomplete authentications. You can also configure the length of time that a tree can be suspended.

You should set this value to the minimum reasonable time required to complete the authentication. For example, if you are sending an email, 10 minutes might be reasonable. The time allowed for suspending authentication must be the same or less than the maximum duration for the tree.

To configure these timeouts, in the AM admin UI, go to **Configure > Authentication > Core Attributes > Trees**. For more information about the properties, see Trees.

Add suspended authentication to custom nodes

You can enable suspended authentication in your custom nodes. For more information, see Action class.

MFA: Web authentication (WebAuthn)

WebAuthn lets users authenticate with an authenticator device, such as the fingerprint scanner on their laptop or phone. The user's browser communicates with the authenticator device.

AM can request the browser to activate the authenticator device with certain criteria; for example:

- The authenticator device must be built-in, not a USB device that the user can eject
- The device must verify the user's identity, not just the user's presence

To use WebAuthn with AM, users must first register their authenticators. If recovery codes are enabled, users should copy their codes on successful registration.

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:

Browsers and platforms supported for WebAuthn

Browser[(1)]	**Platform**
Google Chrome	Desktop
	Android
Safari	Desktop
	iOS
Microsoft Edge	Desktop
Mozilla Firefox	Desktop

[1] Latest stable versions are supported.

Create authentication trees for 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.

This procedure assumes the following:

- The WebAuthn Profile Encryption Service is configured.

  This 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.

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.

To create a multi-factor authentication tree for WebAuthn authentication, and registration if required, perform the following steps:

1. In the AM admin UI, go to **Realms** > ***Realm Name*** > **Authentication** > **Trees**.
2. Create the authentication tree as follows:
   - Click **Create Tree**.
   - Specify a name of your choosing, for example, `myWebAuthnTree` , and 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.
   - Add the following nodes to the authentication tree:
     - Username Collector node

- Password Collector node
- WebAuthn Authentication node
- Data Store Decision node
- WebAuthn Registration node

- Connect the nodes as demonstrated in the following figure:

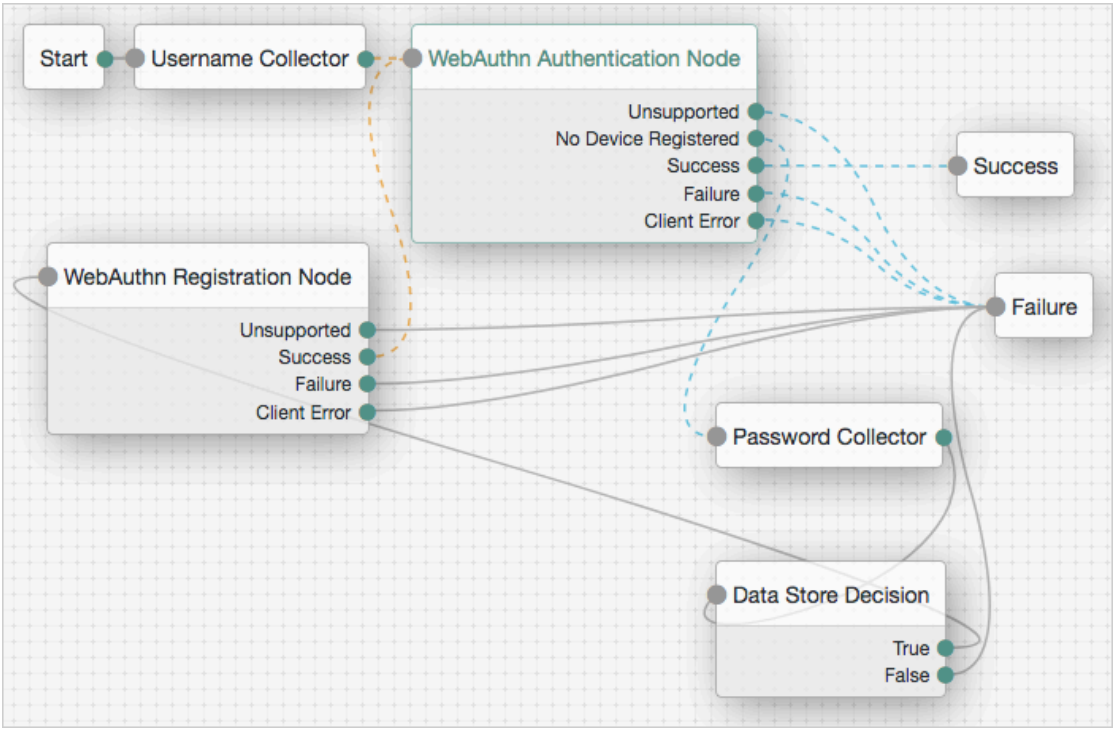


Figure 7. Standalone AM

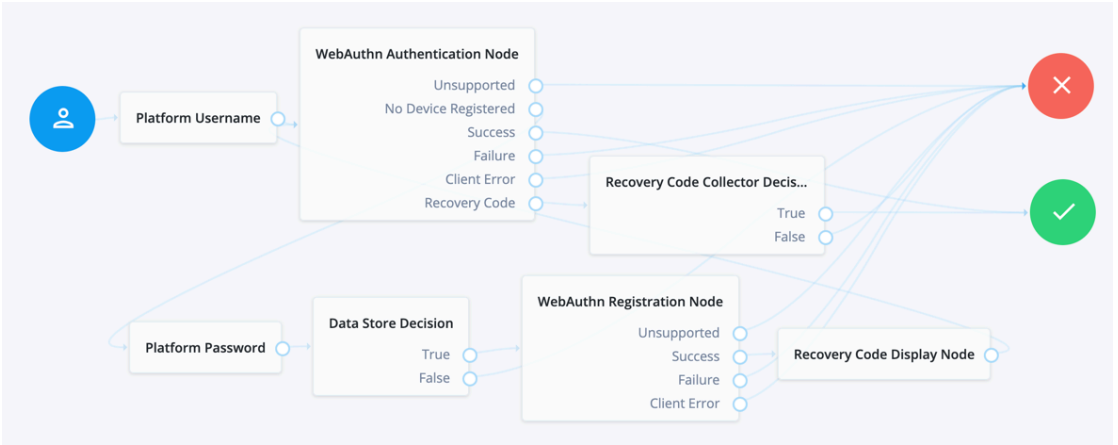


Figure 8. ForgeRock Identity Platform

- Save your changes.

3. Test your WebAuthn authentication and registration tree as follows:

   - Log out of AM, and then go to a URL similar to the following:
     `https://openam.example.com:8443/openam/XUI/?realm=/alpha&service=myWebAuthnTree#login`

IMPORTANT

IMPORTANT

You must connect over HTTPS in order to use Web Authentication.

- Enter the username of an existing account in the specified realm. For example, enter `demo`.
- If the `demo` user does not have a registered device:
  - When asked for the user's password, enter the default `Ch4ng31t`.
  - At the following screen, register a WebAuthn authenticator by performing an authorization gesture, for example press the button on a connected Yubikey.

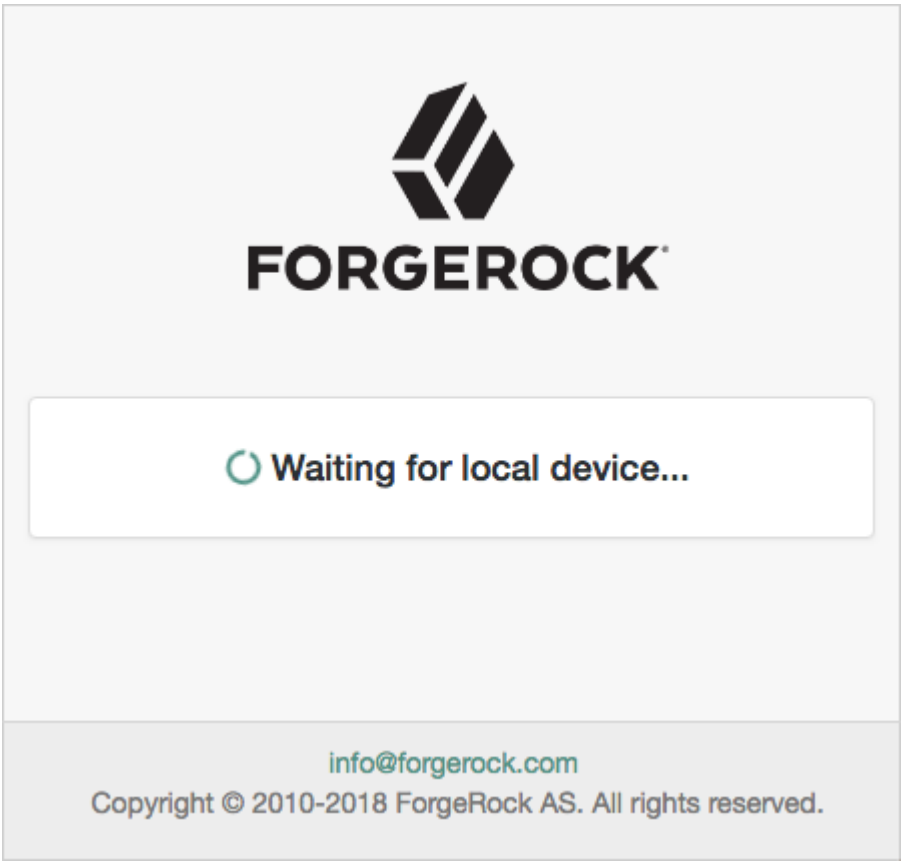


Figure 9. The WebAuthn Registration node waiting for an authenticator (Standalone AM)

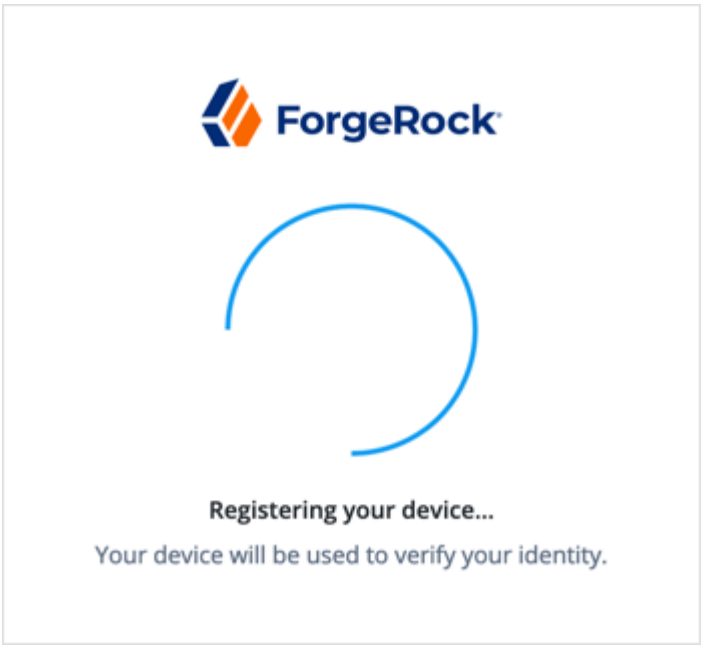


Figure 10. The WebAuthn Registration node waiting for an authenticator (ForgeRock Identity Platform)

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.

- 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`.

Configure usernameless authentication with ForgeRock Go

With ForgeRock Go, you can create a secure and seamless login experience by authenticating with any credential on the user's device that supports FIDO2 WebAuthn.

You can also extend passwordless authentication to include usernameless authentication with popular authenticators that support resident keys; for example, Windows Hello (biometric authenticators).

To use usernameless authentication, you must register an authenticator that supports resident keys to the user's profile, and enable the option to associate a certificate on the

device with the user's username.

Once registered, that device can be used to authenticate the user without them having to provide their credentials; they just have to select the appropriate entry to use from the list their device provides.

To configure usernameless authentication with ForgeRock Go, create a Web Authentication registration tree to associate a device that supports resident keys with a user. The registration tree is similar to that described in Create authentication trees for WebAuthn.

Create a second tree that lets users authenticate to AM without entering their username or password, by using Forgerock Go.

NOTE

The trees in this procedure are examples, and do not provide user-friendly features, such as allowing retries, or redirecting to further help on failures.

1. In the AM admin UI, select the realm that will contain the ForgeRock Go registration tree.
2. Create the registration tree as follows:
    - Select **Authentication > Trees**, and click **Create Tree**.
    - Enter a name, for example, `fr-go-reg`, and 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 side and dropping it into the designer area.
    - Add the following nodes to the authentication tree:
        - Username Collector node
        - Password Collector node
        - WebAuthn Authentication node
        - Data Store Decision node
        - (Optional) Scripted Decision node

          When configured for ForgeRock Go, the WebAuthn Registration node will store the value of the `username` authentication tree shared state variable in the device by default. This value will later be used to identify the user during authentication.

          Use a Scripted Decision Node to customize the display name or string to be saved in the shared state. You will later configure the variable containing the data in the WebAuthn Registration node.

▼ *Example JavaScript to create display names*

```
var username = sharedState.get("username");
var displayName = '';

var fullName = idRepository.getAttribute(username,
"CN").iterator().next();
var email = idRepository.getAttribute(username,
"mail").iterator().next();

if(fullName){
    displayName += fullName;
}

if(email){
    displayName += ' (' + email + ')';
}

sharedState.put("displayName",
displayName.toString());
outcome = "continue";
```

- (Optional) Page node

- Connect the nodes as demonstrated in the following figure:

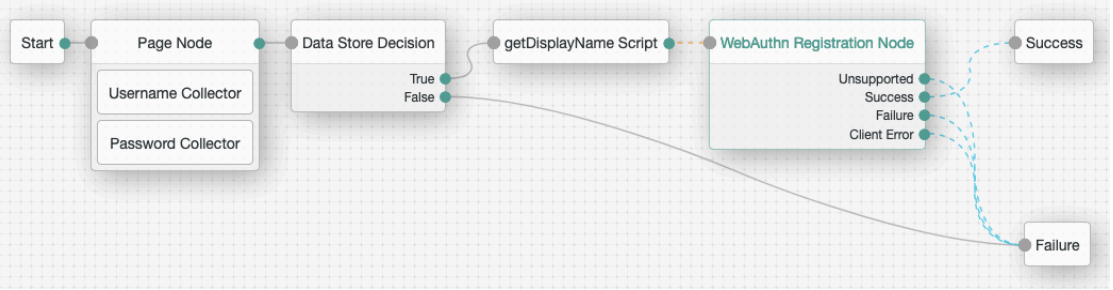
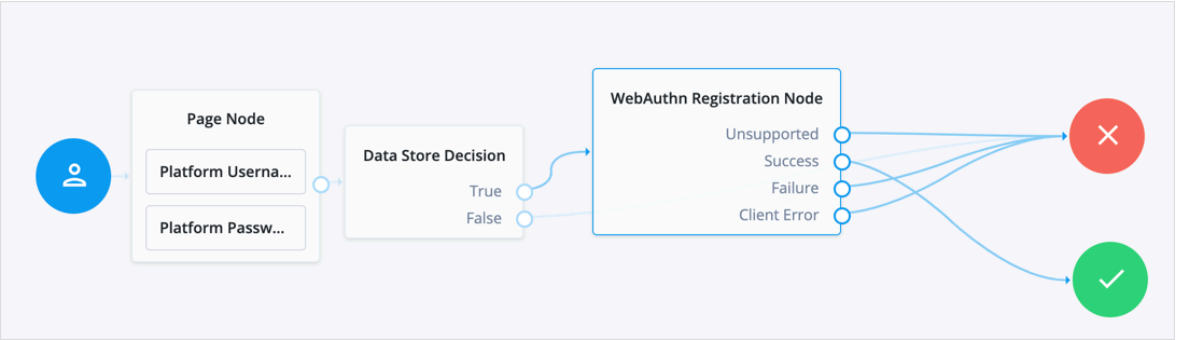


Figure 11. Standalone AM

3. ForgeRock Identity Platform



- In the WebAuthn Registration node properties, ensure **Username to device** is enabled.

- (Optional) If you are using a Scripted Decision node to create the display name, enter the shared state variable name into the `Shared state attribute for display name` property in the WebAuthn Registration node.
- (Optional)

  (ForgeRock Identity Platform deployments)

  If you are `not` using the Scripted Decision node to create the display name, enter `userName` into the `Shared state attribute for display name` property in the WebAuthn Registration node.
- Save your changes.

4. Create an authentication tree for ForgeRock Go, and specify a name of your choosing; for example, `fr-go-auth`.
   - Add a WebAuthn Authentication node to the authentication tree.
   - Connect the nodes as demonstrated in the following figure:

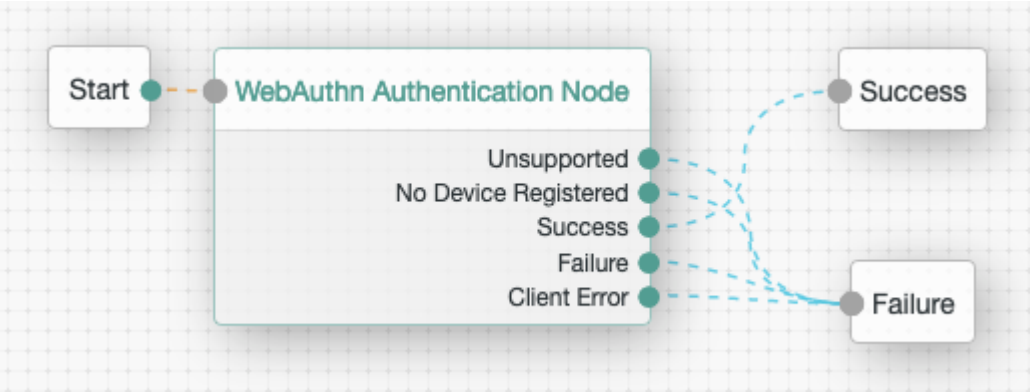


Figure 12. Standalone AM

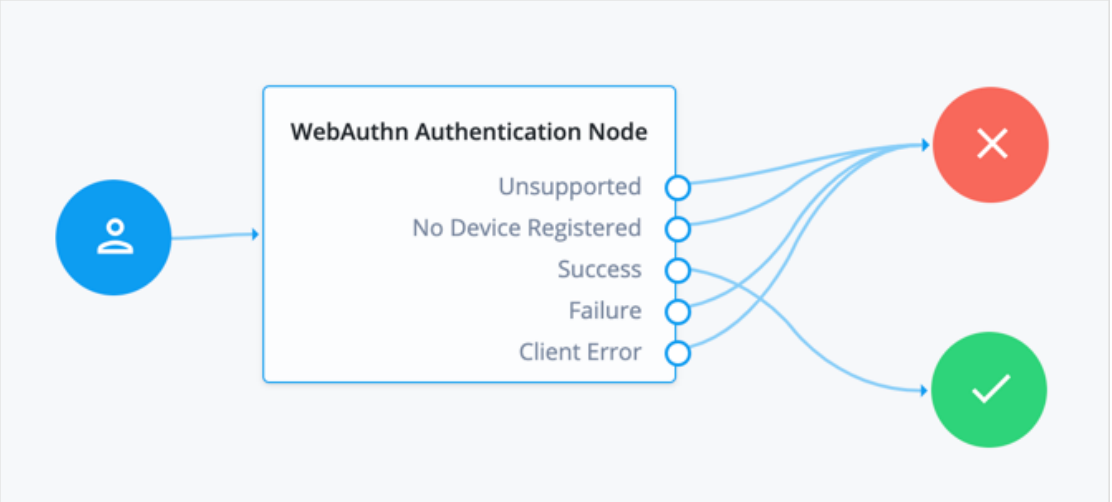


Figure 13. ForgeRock Identity Platform

   - In the WebAuthn Authentication node properties, ensure **Username from device** is enabled.
   - Save your changes.

5. You are now ready to register a device, and authenticate by using ForgeRock Go.

Proceed to Register and authenticate with ForgeRock Go.

Register and authenticate with ForgeRock Go

Follow these steps to register a device for use with usernameless authentication, and then authenticate without having to provide your username or password.

1. To register a device for use with ForgeRock Go:
   - Log out of AM, and then go to your ForgeRock Go registration tree, with a URL similar to the following: `https://openam.example.com:8443/openam/XUI/?realm=/alpha&service=fr-go-reg#login`

     IMPORTANT

     You must connect over HTTPS in order to use Web Authentication.

   - Enter the username and password of an existing account in the specified realm. For example, enter `demo`, and the password `Ch4ng31t`, and click **[Log In]**.
   - If you are authenticating from a FIDO2-enabled device, a dialog will display asking you to choose the method to verify your identity; for example, a USB security key, or built-in biometric sensor.

     Select the option you want to associate with the user.
   - Perform the authorization gesture of the chosen option when asked to do so. For example, scan your fingerprint with TouchID, or press the button on your USB security key.

     If successful, you are taken to the profile page for the user.
   - The new device appears on the Dashboard page, as **New Security Key**.

     Give a suitable name to the device; for example, *Apple Mac TouchID*, by clicking the context icon, ⋮, and selecting **Settings**.
2. To use a device to authenticate without username or password by using ForgeRock Go:
   - Log out of AM, and then go to your ForgeRock Go authentication tree, with a URL similar to the following: `https://openam.example.com:8443/openam/XUI/?realm=/alpha&service=fr-go-auth#login`

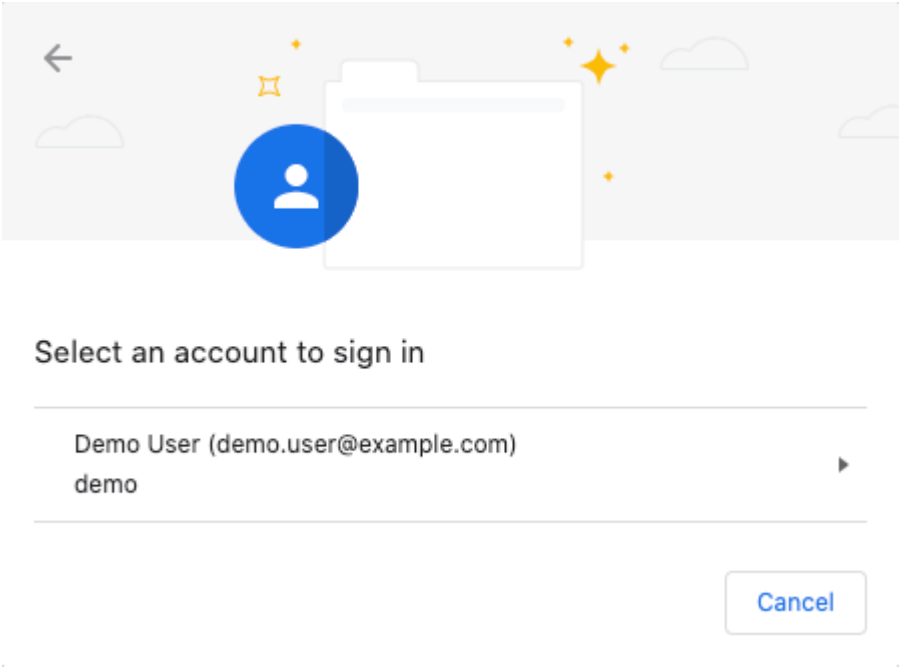
     IMPORTANT

     You must connect over HTTPS in order to use Web Authentication.

   - Perform the authorization gesture of the chosen option when asked to do so. For example, scan your fingerprint with TouchID, or press the button on your

USB security key.

If successful, a list of the accounts associated with the authentication device displays:



Note that in this example the user's full name and email address appear, which were gathered by the Scripted Decision node from the user's profile during registration.

- Select the account that you want to sign in.

  If successful, you are taken to the profile page for the user, without having to enter username or password credentials!

Configure WebAuthn trust anchors

AM 7 added support for a new *CA* attestation type, whereby the attestation data received from a device can be verified as authentic by using the relevant CA certificates.

If the trust chains defined by the CA certificates have CRL or OCSP entries, AM is also able to check for revocation.

To configure trust anchors in AM, you should obtain the CA-issued certificate chains for the devices you intend to verify, and make them available to AM in a secret store.

When the relevant certificate chains are in place, configure the WebAuthn Registration node with the alias of the secret store, and set **Preferred mode of attestation** to either `DIRECT` or `INDIRECT`.

Perform the following steps to enable trust anchors and achieve `CA` attestation for trusted devices:

1. Obtain the CA-issued certificate chains for the devices you want to verify.

   You may need to consult the device manufacturer to obtain the certificate chains.

2. Import the certificate chains into a keystore:

   ```
   $ keytool -import \
     -file /Downloads/vendor-a-ca.crt \
     -alias "vendor-a-devices" \
     -storetype JCEKS \
     -storepass changeit \
     -keystore
   /path/to/openam/security/keystores/webauthnTrustStore.jceks
   ```

   The command above imports a hypothetical trust chain from "Vendor A" into a secret store named `webauthnTrustStore.jceks`, located in the default AM path for keystores, `/path/to/openam/security/keystores`.

   If the keystore does not exist, the command creates it and sets the store password to `changeit`; otherwise, it adds the specified certificate to the secret store.

3. Repeat the previous step until the `webauthnTrustStore.jceks` secret store contains all of the CA-issued certificate chains for the devices you want to verify.

4. Ensure that the password to access the new `webauthnTrustStore.jceks` secret store is available to your AM instance; for example, by encrypting the password and adding the result to a new file named `webauthnStorepass`, in `/path/to/openam/security/secrets/encrypted`.

   For more information, see File System Secret Volumes Secret Stores.

5. In the AM admin UI, go to the realm containing the web authentication registration tree that will attempt CA-level attestation, go to Authentication > Trees, and click your registration tree.

6. Select the WebAuthn Registration Node, and in the properties pane:

   - Set the **Preferred mode of attestation** property to `DIRECT` or `INDIRECT`.
   - Set the **Trust Store alias** property to a string that will identify both the name of the trust store, and will be the suffix of the secret ID used for mappings; for example, `webauthnTrustStore`.
   - If you want to act upon the attestation type achieved when registering a device; for example, using a script, then you should enable the **Store data in transient state** property.

     When this is enabled, the WebAuthn Registration Node stores the level of attestation achieved in a variable named `webauthnAttestationType`, in the transient state of the tree.

     TIP

Use code similar to the following JavaScript to read the value of `webauthnAttestationType` :

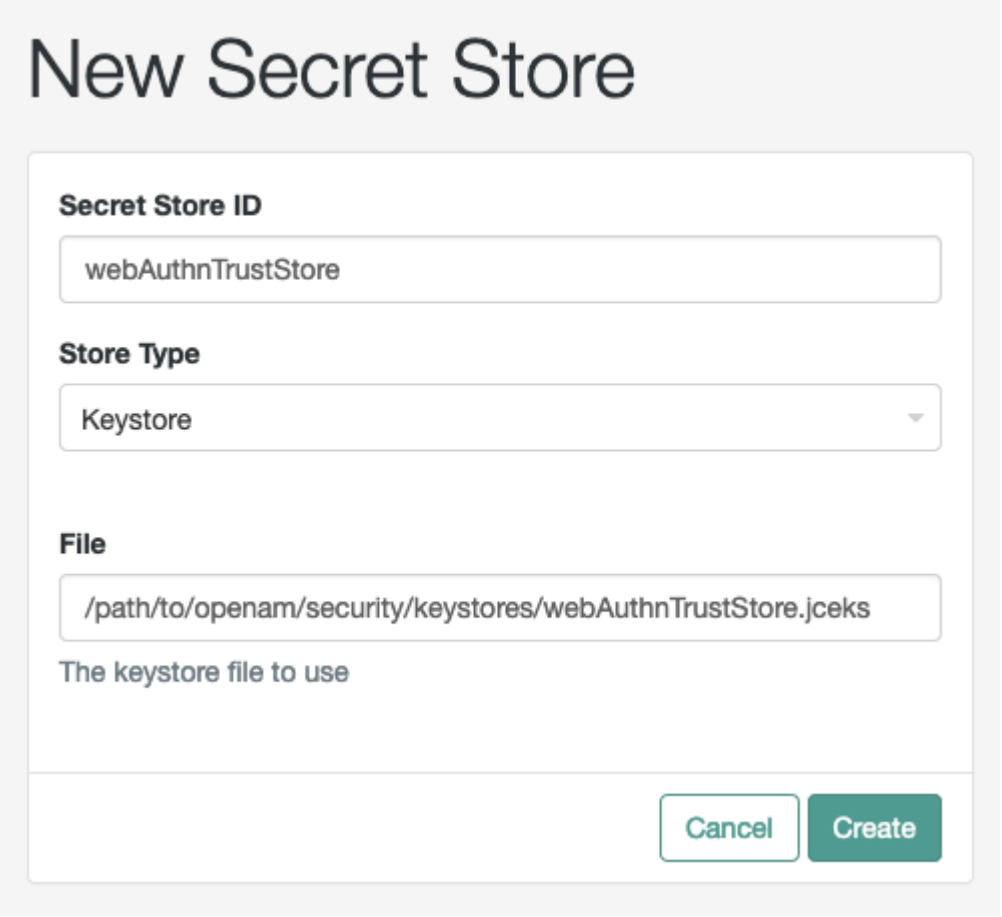
```
var attestationLevel =
transientState.get("webauthnAttestationType");
```

- Save your changes.

  For more information on the available properties, see WebAuthn Registration node.

7. Navigate back to the realm page, select **Secret Stores**, and click **[Add Secret Store]**.
8. In **Secret Store ID**, enter the alias you specified in the registration node earlier; for example, `webauthnTrustStore` .

   Select the store type, specify the path to the store, and click **[Create]**.



9. Set the **Store password secret ID** to the name of the file you created earlier with the encrypted value of the store password in; for example, `webauthnStorepass` .

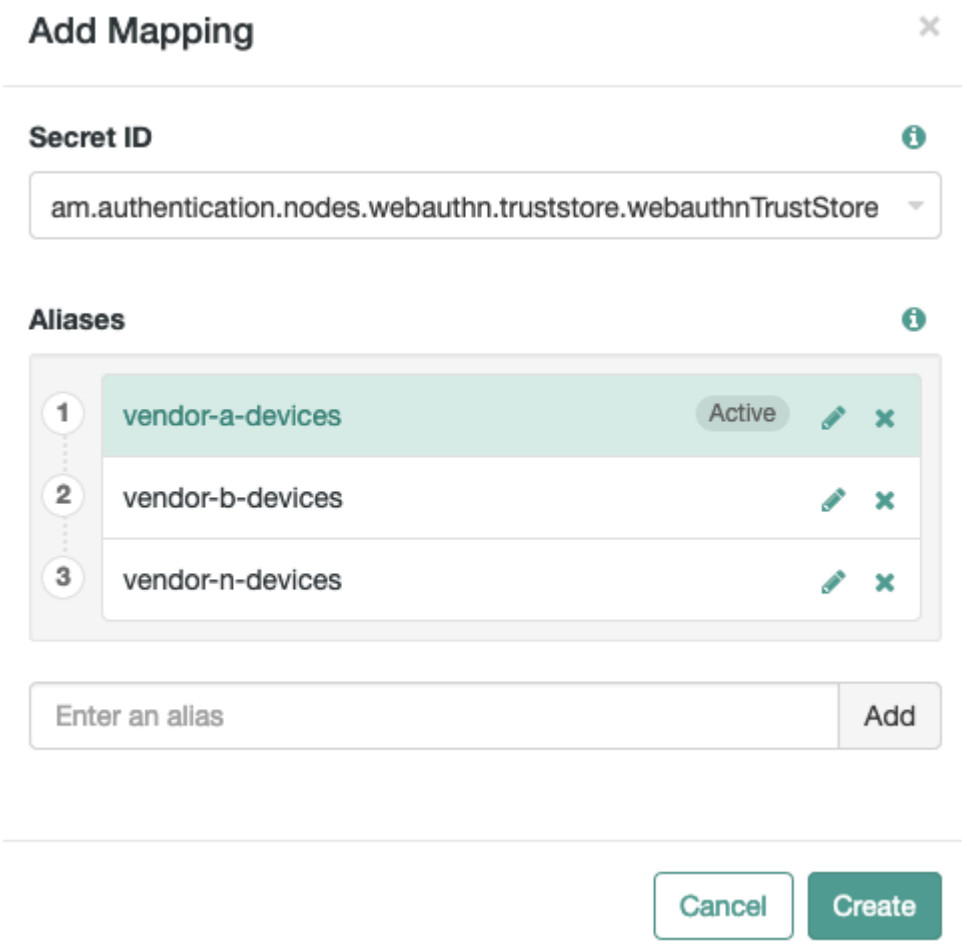
   Save your changes.
10. On the **Mappings** tab, click **[Add Mapping]**.

11. In **Secret ID**, select the ID that begins with `am.authentication.nodes.webauthn.truststore`, and has the alias you specified earlier as the suffix.

    For example, `am.authentication.nodes.webauthn.truststore.webauthnTrustStore`.

12. Enter the alias of the certificate chains you want to use for verification, and click **[Add]**.

    Repeat this step to add all the aliases of certificate chains you want to use for CA-level attestation:



13. Save your changes.

    Your registration tree is now ready to verify the attestation data against the list of configured certificate chains.

MFA: 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.

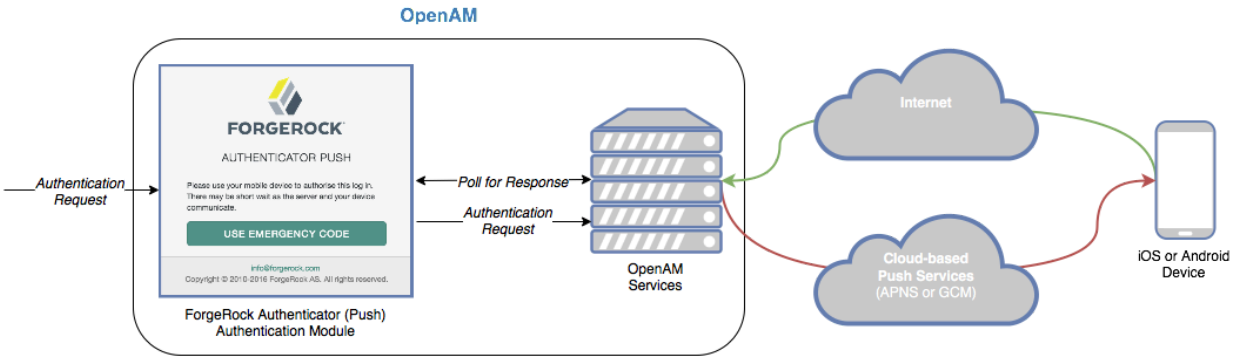


Figure 14. Overview of Push Authentication

The following steps occur when AM receives an authentication request and is configured for MFA 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 Manage devices for MFA.
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.

   AM begins to poll 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 Test push authentication.

The app returns the response to the AM site.

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.

   AM detects the accepted record and redirects the user to their profile page, completing the authentication.

The following table summarizes the tasks you need to perform to implement Push authentication in your environment:

Task	Resources
Configure authentication Depending on your environment, choose whether to configure Push authentication on trees or chains. ForgeRock recommends that you implement authentication trees. If you are planning to implement passwordless push authentication, see also Limitations when using passwordless push authentication.	• Create trees for push authentication and registration • Chains for push authentication
Test push authentication After configuring AM, download the ForgeRock Authenticator app and test your configuration.	• Test push authentication

Create trees for push authentication and registration

Push authentication uses authentication trees to receive push notifications and to perform the actual authentication itself.

Authentication trees can be used for passwordless authentication using push notifications. When configured for passwordless authentication, the authentication flow 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.

Before implementing passwordless push authentication, consider the Limitations when using passwordless push authentication.

Create a tree for push authentication

The procedure assumes the following:

- Users will provide user IDs and passwords as the first step of MFA.
- A push notification will be sent to the device as a second factor to complete authentication.
- The following services are configured:

  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.

  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.

To create an MFA tree, follow these steps:

1. In the AM admin UI, go to **Realms** > ***Realm Name*** > **Authentication** > **Trees**, and create the authentication tree as follows:
   - Select **Authentication** > **Trees**, and click **Create Tree**.
   - Specify a name of your choosing; for example `myPushTree`, and 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.
   - Add the following nodes to the authentication tree:
     - Username Collector node
     - Password Collector node
     - Push Sender node
     - Push Result Verifier node

- Polling Wait node
- MFA Registration Options node
- Opt-out Multi-Factor Authentication node
- Push Registration node
- Get Authenticator App node
- `Success` Node

- Connect the nodes as demonstrated in the following figure:

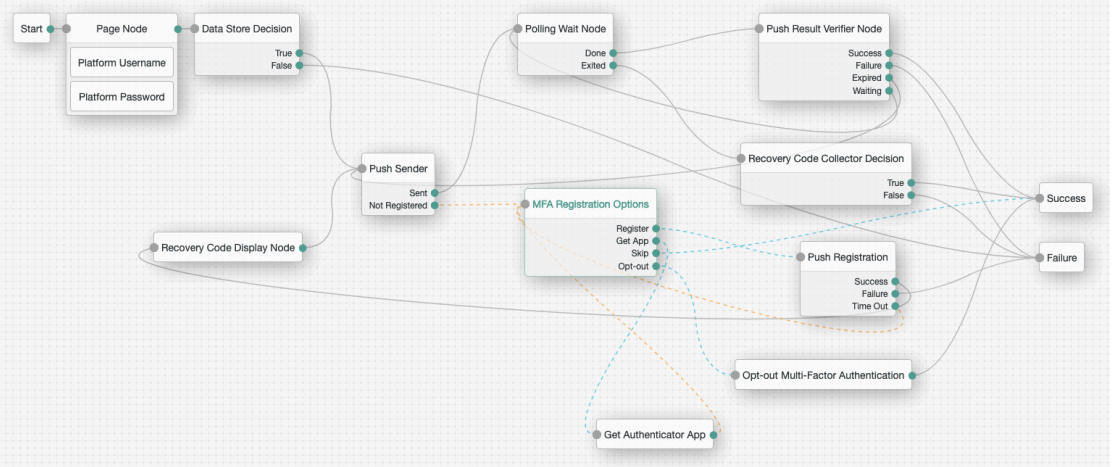


Figure 15. Example Push Tree (Standalone AM)

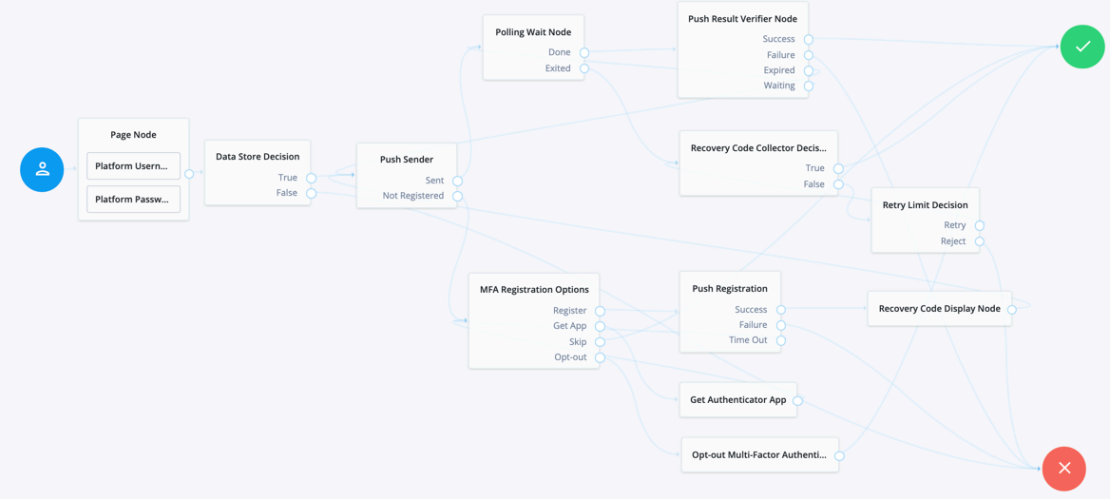


Figure 16. Example Push Tree (ForgeRock Identity Platform)

- Save your changes.

2. Test your authentication tree as follows:

- Log out of AM, and then go to a URL similar to the following:
  `https://openam.example.com:8443/openam/XUI/?realm=/alpha&service=`*`myPushTree`*`#login`
- Follow the procedure described in Test push authentication to verify that you can use the ForgeRock Authenticator app to perform MFA. If the authentication tree is correctly configured, authentication is successful and AM displays the user profile page.

Create a tree for passwordless authentication

The procedure assumes the following:

- Users will provide only their user IDs as the first step of MFA.
- This procedure assumes users have a device registered 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.
- The following services are configured:

  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.

  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.

To create an MFA tree for passwordless authentication, follow these steps:

1. In the AM admin UI, go to **Realms** > ***Realm Name*** > **Authentication** > **Trees**, and create the authentication tree as follows:
   - Select **Authentication** > **Trees**, and click **Create Tree**.
   - Enter a name for the tree, for example `myPasswordlessAuthTree` , and 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.
   - Add the following nodes to the authentication tree:
     - Username Collector node
     - Push Sender node
     - Push Result Verifier node

- Polling Wait node
- `Success` Node

- Connect the nodes as demonstrated in the following figure:

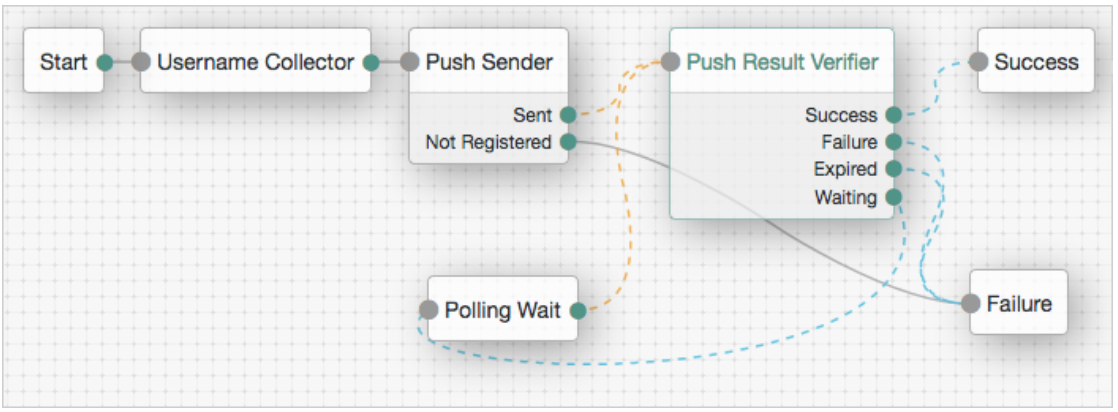


Figure 17. Passwordless Push Authentication Example (Standalone AM)

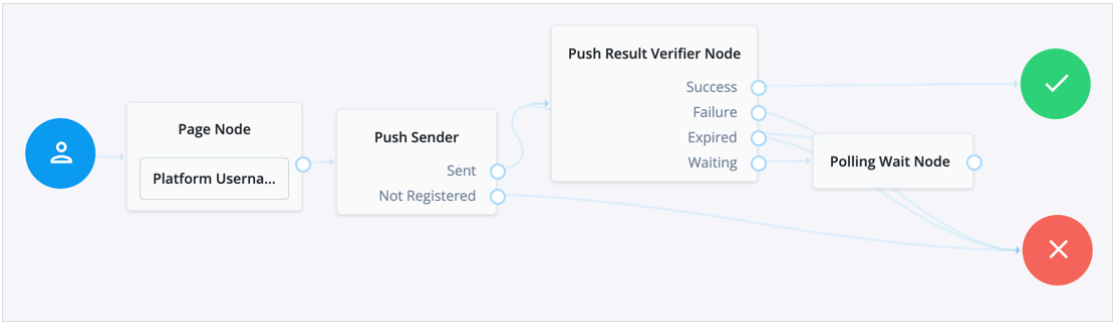


Figure 18. Passwordless Push Authentication Example (ForgeRock Identity Platform)

- Save your changes.

2. Test your authentication tree as follows:
   - Log out of AM, and then go to a URL similar to the following: `https://openam.example.com:8443/openam/XUI/?realm=/alpha&service=myPasswordlessAuthTree#login`
   - Follow the procedure described in Test push authentication to verify that you can use the ForgeRock Authenticator app to perform MFA. If the authentication tree is correctly configured, authentication is successful and AM displays the user profile page, without having to enter a password.

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 Create a 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.

Before implementing passwordless push authentication, consider the Limitations when using passwordless push authentication.

Create a chain for push authentication

The procedure assumes the following:

- Users will provide user IDs and passwords as the first step of MFA.
- 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.
- The following services are configured:

  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.

  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.

To create an MFA chain that uses the ForgeRock Authenticator (Push) Registration and ForgeRock Authenticator (Push) modules, perform the following steps:

1. In the AM admin UI, select the realm that will contain the authentication chain.
2. Create a ForgeRock Authenticator (Push) Registration authentication module as follows:
    - Select **Authentication** > **Modules**, and click **Add Module**.
    - Fill in fields on the **New Module** page as follows:
        - **Name**: Choose a module name, for example *push-reg*.
        - **Type**: Select ForgeRock Authenticator (Push) Registration.
    - Click **Create**.
    - 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.
3. Create a ForgeRock Authenticator (Push) authentication module as follows:
    - Select **Authentication** > **Modules**, and click **Add Module**.
    - Complete the **New Module** page as follows:
        - **Name**: Specify a module name, for example *push-auth*.
        - **Type**: Select ForgeRock Authenticator (Push).
    - Click **Create**.
    - 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.
4. Create the authentication chain as follows:
    - Select **Authentication** > **Chains**, and click **Add Chain**.
    - Enter a name for the chain, for example `myPushAuthChain` , and click **Create**.
    - Add the Data Store authentication module to the authentication chain as follows:
        - Click **Add a Module**.
        - Fill in the New Module dialog box, specifying the Data Store authentication module. For this example, specify the `Requisite` flag.
        - Click **OK**.

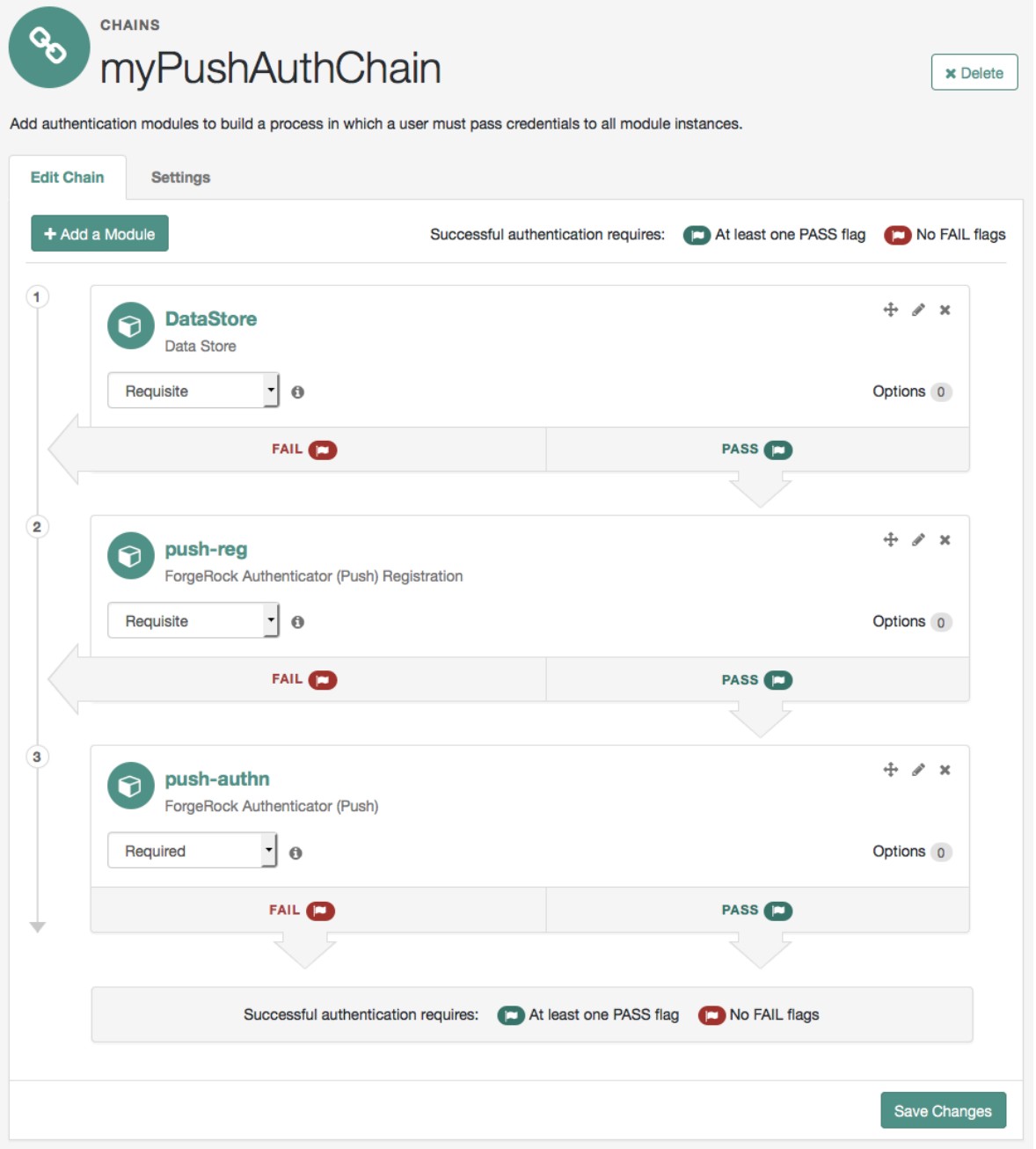
          The graphic showing your authentication chain now includes a Data Store authentication module.
    - Add the ForgeRock Authenticator (Push) Registration authentication module to the authentication chain as follows:
        - Click **Add a Module**.

- Complete the **New Module** dialog box, specifying the ForgeRock Authenticator (Push) Registration authentication module that you just created. For this example, specify the `Requisite` flag.
- Click **OK**.

  The graphic showing your authentication chain now includes a Data Store, and a ForgeRock Authenticator (Push) Registration authentication module.

- Add the ForgeRock Authenticator (Push) authentication module to the authentication chain as follows:
  - Click **Add a Module**.
  - Complete the **New Module** dialog box, specifying the ForgeRock Authenticator (Push) authentication module that you created. For this example, specify the `Required` flag.
  - 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.



- Save your changes.

5. Test your authentication chain as follows:
   - Log out of AM, and then go to a URL similar to the following:
     `https://openam.example.com:8443/openam/XUI/?realm=/&service=myPushAuthChain#login`
   - Follow the procedure described in Test push authentication to verify that you can use the ForgeRock Authenticator app to perform MFA. If the chain is correctly configured, authentication is successful and AM displays the user profile page.

Create a chain for push registration and passwordless authentication

The procedure assumes the following:

- Users will provide only their user IDs as the first step of MFA.
- 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 Create a 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.
- The following services are configured:

  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.

  Push Notification Service
  Configures how AM sends push notifications to registered devices, including endpoints, and access credentials.

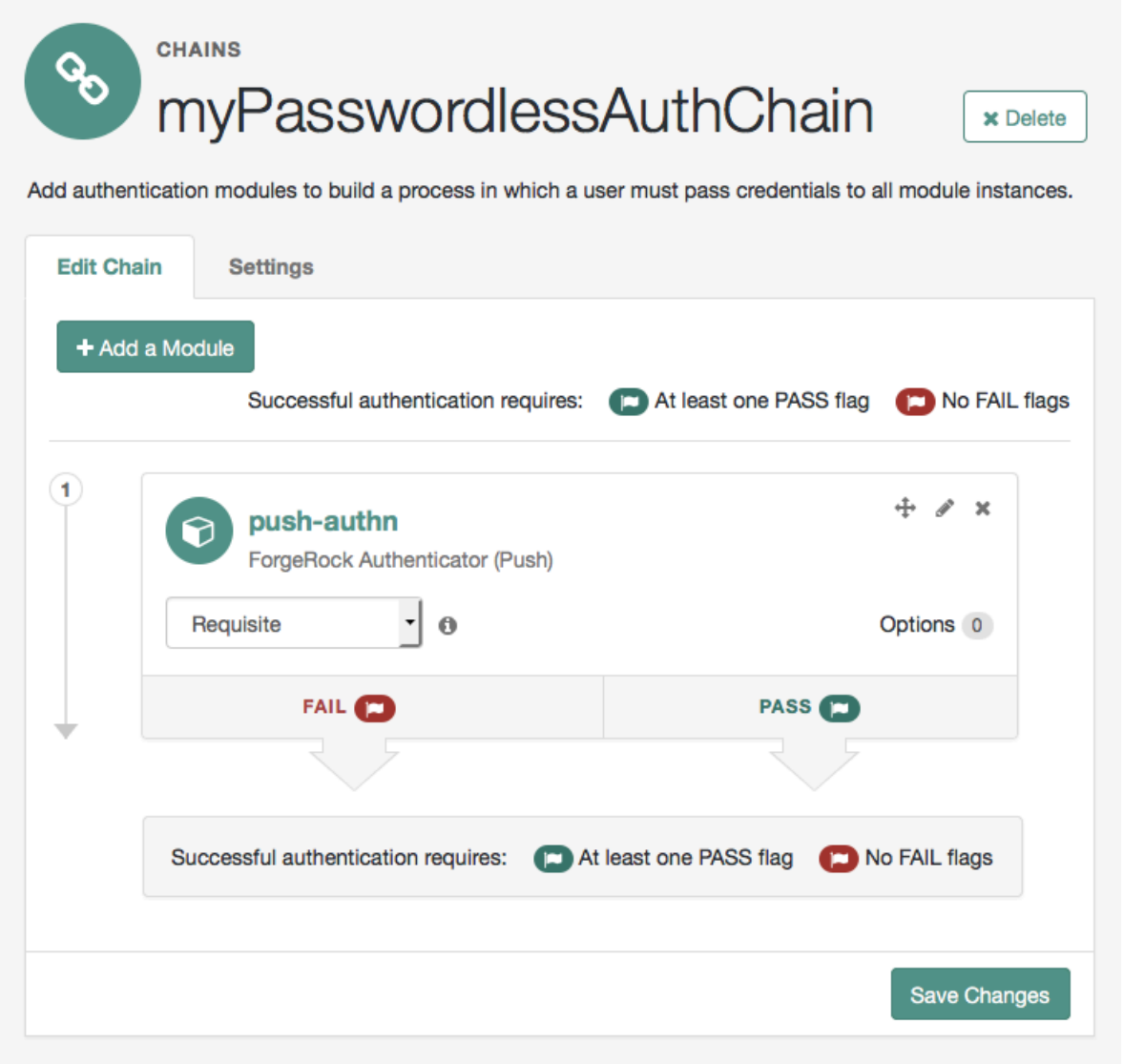
  For information on provisioning the credentials required by the Push Notification Service, see How To Configure Service Credentials (Push Auth, Docker) in Backstage in the *ForgeRock Knowledge Base*.

  For detailed information about the available properties, see Push Notification Service.

To create an MFA chain that uses the ForgeRock Authenticator (Push) module for passwordless authentication, perform the following steps:

1. In the AM admin UI, select the realm that will contain the authentication chain.
2. Create the authentication chain as follows:
   - Select **Authentication > Chains**, and click **Add Chain**.
   - Specify a name, for example *myPasswordlessAuthChain*, and click **Create**.
   - Add the ForgeRock Authenticator (Push) authentication module to the authentication chain as follows:
     - Click **Add a Module**.
     - Complete the **New Module** dialog box, specifying the ForgeRock Authenticator (Push) authentication module that you created. For this example, specify the `Requisite` flag.
     - Click **OK**.

       The graphic showing your authentication chain now includes a ForgeRock Authenticator (Push) authentication module.



- Save your changes.

3. Test your authentication chain as follows:
   - Log out of AM, and then go to a URL similar to the following:
     `https://openam.example.com:8443/openam/XUI/?realm=/#login/&service=myPasswordlessAuthChain`
   - Follow the procedure described in Test push authentication to verify that you can use the ForgeRock Authenticator app to perform MFA. If the chain is correctly configured, authentication is successful and AM displays the user profile page, without having to enter a password.

Test 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 sends a push notification to the registered device.

If the user does not yet have a device registered for push authentication, refer to Register.

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.

Receive push notifications

On your registered device, you receive a push notification from AM. Depending on the state of the device and the ForgeRock Authenticator application, respond to the notification as follows:

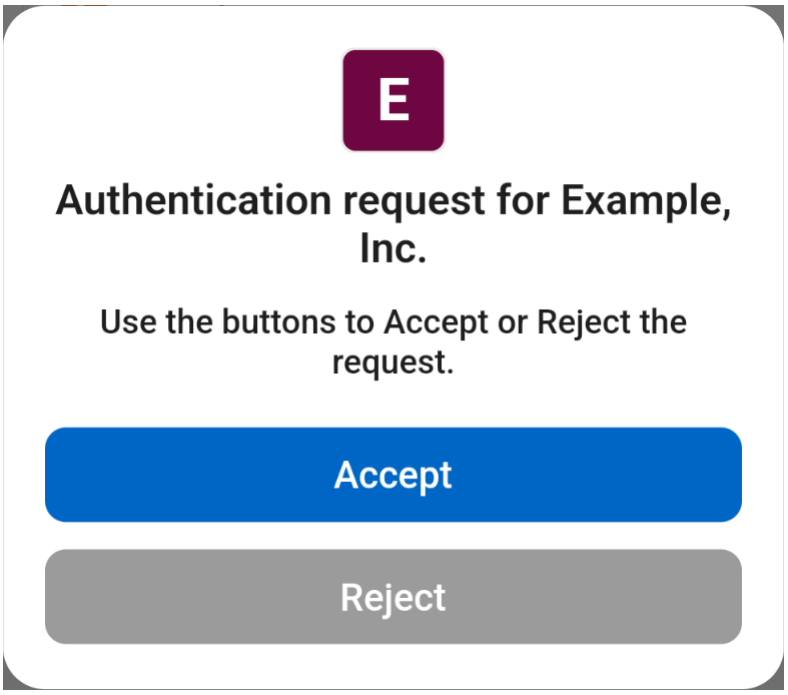
- Unlock the device, if necessary, when you receive a device notification from the application.

  The ForgeRock Authenticator application opens and displays the push notification.
- If the device is unlocked, but the ForgeRock Authenticator application is not open, select the device notification to open the application and display the push notification.
- Open the ForgeRock Authenticator application to respond quickly to notifications.

Approve requests

How you approve requests depends on the ForgeRock Authenticator application settings, and on what the device supports.

Default settings for push notifications use a simple pop up in the application, similar to the following:



Deny requests

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 **Reject** button.

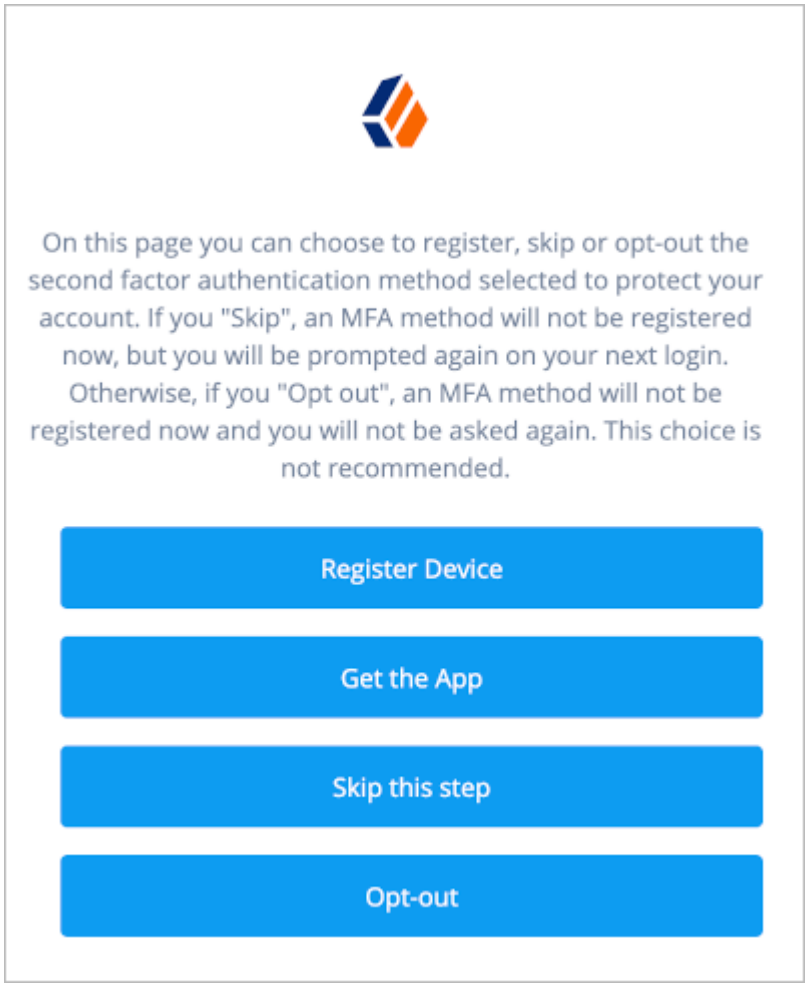
NOTE

If you do not approve or deny the request on the registered device, the AM Push Authentication page times out and authentication fails.

You can configure this through the **Message Timeout** in the Push Sender node for the tree.

Register

If your credentials are valid but your profile is missing the metadata for a registered device registered, the MFA Registration Options node of the tree governs what happens:



Register Device

Configure the journey to continue to the Push Registration node.

When completing the journey, scan the QR code it displays with the ForgeRock Authenticator application.

For additional details, refer to Register the ForgeRock Authenticator for multi-factor authentication.

Get the application
Configure the journey to continue to the Get Authenticator App node.

When completing the journey, follow the link needed to obtain the ForgeRock Authenticator application for your device.

Skip this step
Displayed only if the node configuration allows the user to skip. (Optional) In the example journey, skipping is linked to the Success node.

Opt-out
Configure the journey to continue to the Opt-out Multi-Factor Authentication node and let the user not use push.

In the example journey, opting out is linked to the Success node.

Configure successful registration to return to the Push Sender node, which starts the actual push notification stage of the journey, and the user can receive push notifications.

Limitations when using passwordless push authentication

When authenticating to a passwordless push authentication tree or 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 MFA, and not on their own.

MFA: 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 (OTP) 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 Reset registered devices over 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 between authentication modules that support HOTP

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.

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	✔	✔
AM can generate a HOTP password and send it to end users in a text message or an e-mail	✖	✔
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	✔	✖
End users can authenticate with a TOTP password	✔	✖

Requirement	Available With the ForgeRock Authenticator (OATH) Authentication Module?	Available With the HOTP Authentication Module?
End users can opt out of providing a one-time password	✔	✖
End users can authenticate using XUI	✔	✔

One-time password authentication using trees

This section describes how to create and configure trees for one-time password authentication.

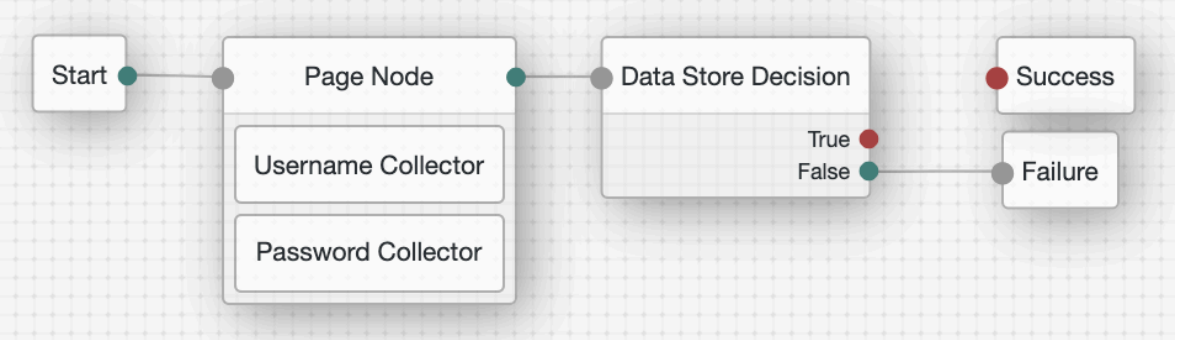
Create a tree for one-time password authentication

To create an example authentication tree that uses OATH authentication, perform the following steps:

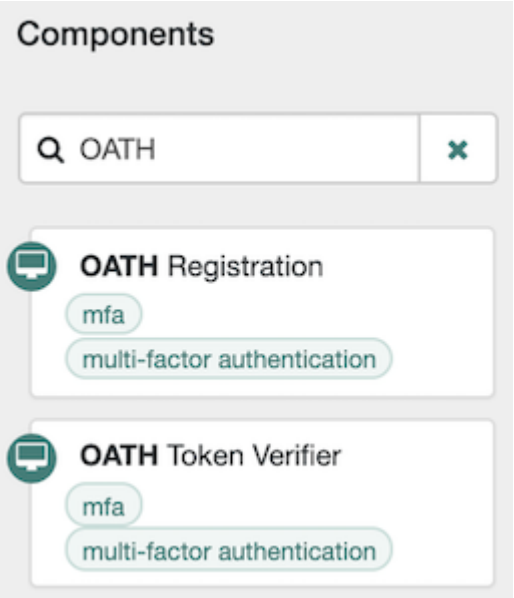
1. In the AM admin UI, select the realm that will contain the authentication tree.
2. Select **Authentication > Trees**, and click **+Create Tree**.
3. Type a name for your tree in the **New Tree** page; for example, `myAuthTree` , and click **Create**.

   The authentication tree designer page is displayed with default Start, Failure, and Success nodes.

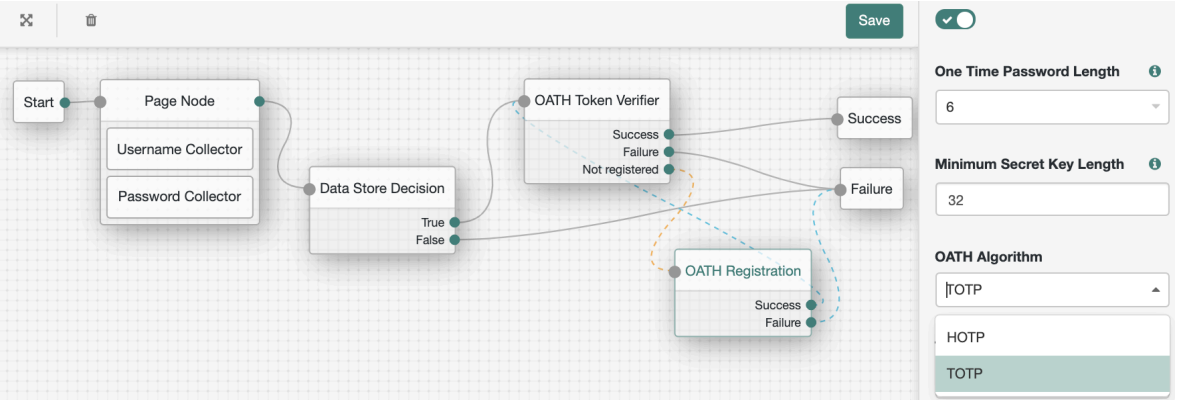
   For information about using the authentication tree designer, see Create an authentication tree.
4. Add the following nodes to the designer area:
   - Page node
   - Username Collector node
   - Password Collector node
   - Data Store Decision node
5. Connect the nodes as shown:



6. Type 'OATH' to filter the list of nodes in the Components panel box:



7. Drag an OATH Token Verifier node and an OATH Registration node onto the designer area.
8. For both OATH nodes, set the OATH Algorithm property to TOTP, and connect to the existing nodes as follows:



   The value for OATH Algorithm must be the same for both nodes. For this example, select TOTP to generate a new OTP at a specified time step interval.

9. Save your changes.

   Note that the tree you have created is a simple example for the purposes of demonstrating a basic OATH authentication journey. In a production environment, you could include additional nodes, such as:

MFA Registration Options node

Provides options for users to register a multi-factor authentication device, get the authenticator app, or skip the registration process.

Opt-out Multi-Factor Authentication node

Sets an attribute in the user's profile which lets them skip multi-factor authentication.

Recovery Code Display node

Lets a user to view recovery codes to use in case they have lost or damaged their registered authenticator device.

Retry Limit Decision node

Lets a journey loop a specified number of times, for example, to allow a user to retry entering their OATH token.

For information about how to configure these nodes, see Authentication nodes configuration reference.

10. Test your authentication tree as follows:
    - Log out of AM, and then go to a URL similar to the following: `https://openam.example.com:8443/openam/XUI/?realm=alpha&service=myAuthTree#login`
    - Log in using the username and password. For example, enter `demo`, and the password `Ch4ng31t`.
    - On successful login, if the screen displays a QR code, you will need to register your device.

      To register the device with the ForgeRock Authenticator, follow the instructions as described in Register the ForgeRock Authenticator for MFA.
    - Follow the procedure described in Authenticate with a one-time password to verify that you can authenticate using the ForgeRock Authenticator app.

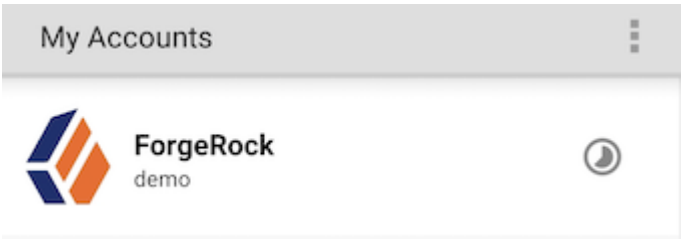
Authenticate with a one-time password

This example task assumes the following prerequisites:

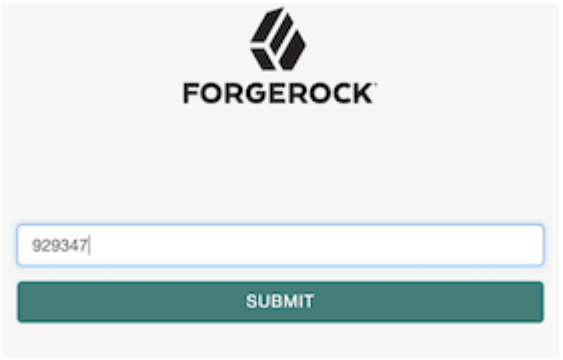
- The authentication tree is set up as described in Create a tree for one-time password authentication.
- You have successfully logged in with valid credentials.
- You have registered your device for ForgeRock Authenticator (OATH) authentication.

Follow these steps to complete one-time password (OTP) authentication:

1. On your registered device, open the ForgeRock Authenticator app, and then tap the OTP section for the account matching the user ID. For example:



2. Note the OTP that is displayed on the screen. This is automatically refreshed at an interval defined in the OATH node configuration. If the animated timer indicates the OTP is close to expiry, wait until a new OTP is generated.
3. On the ForgeRock Authenticator (OATH) page in AM, enter the OTP that the authenticator app generated on your phone, and click Submit:



AM displays the user's profile page.

One-time password authentication using chains

This section covers one-time password authentication.

Create a 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 Authentication modules and chains for details about authentication module flag settings.

- Users can opt out of one-time password authentication.
- The ForgeRock Authenticator (OATH) service is configured.

  This service specifies the attribute in which to store information about the registered OATH device, and whether to encrypt that information. It 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.

To create a multi-factor authentication chain that uses the ForgeRock Authenticator (OATH) module, perform the following steps:

1. In the AM admin UI, select the realm that will contain the authentication chain.
2. You can allow users to opt out of using OATH-based one-time passwords as follows:
    - Select **Authentication > Settings > General**.
    - Make sure that **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 Let users opt out of one-time password authentication (OATH).
3. Create a ForgeRock Authenticator (OATH) authentication module as follows:
    - **Select Authentication > Modules**, and click **Add Module**.
    - Complete the **New Module** page as follows:
        - Name: Specify a module name.
        - Type: Select ForgeRock Authenticator (OATH).
    - Click **Create**.
    - Configure the ForgeRock Authenticator authentication module to meet your organization's requirements.

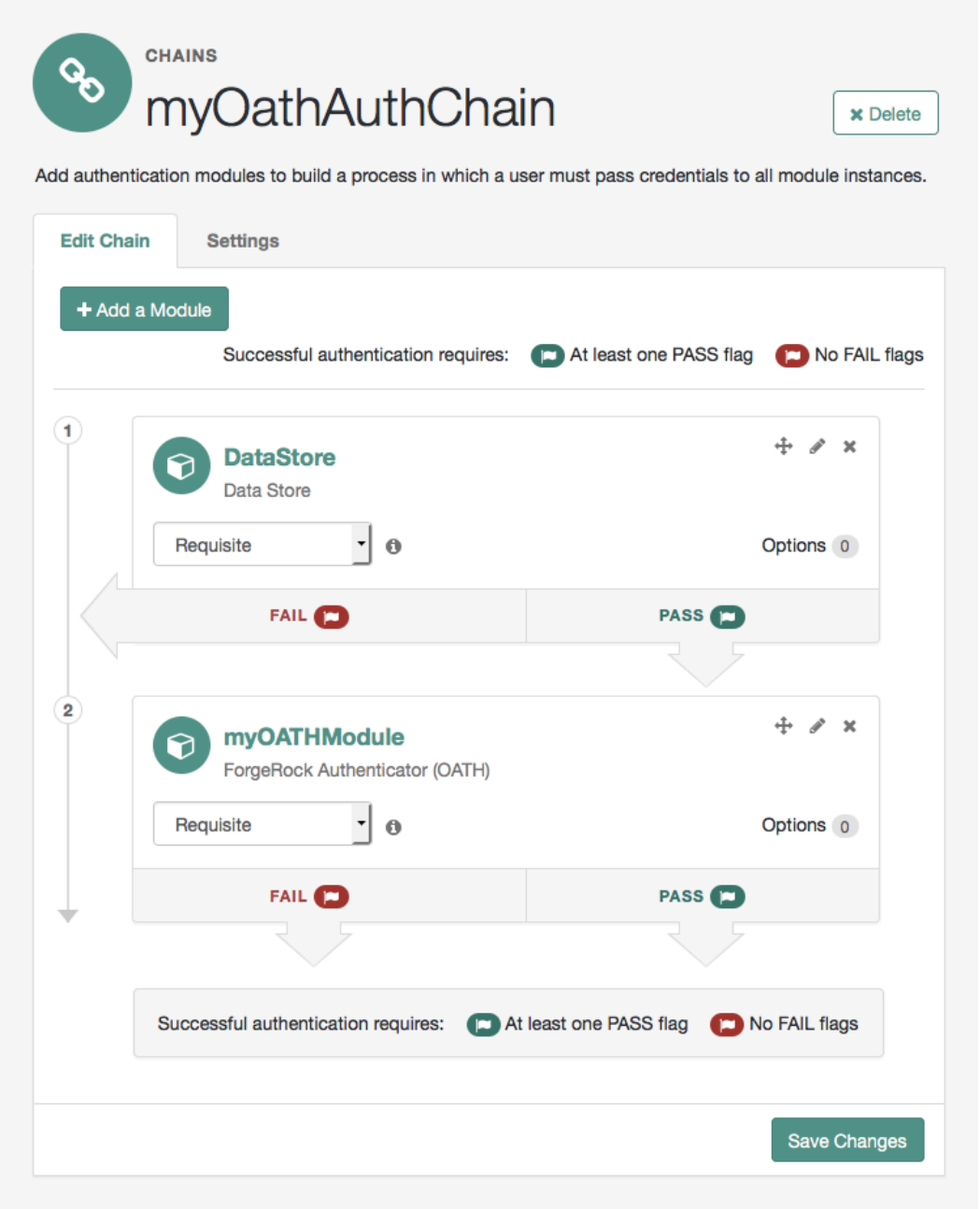
      For more information about the authentication module's configuration settings, see ForgeRock Authenticator (OATH) Authentication Module.
4. Create the authentication chain as follows:
    - Go to **Authentication > Chains**, and click **Add Chain**.
    - Specify a name, for example *myOATHAuthChain*, and click **Create**.
    - Click **Add a Module**. Complete 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 Authentication modules and chains for information about authentication module flag settings.
    - Click **OK**.

      A graphic showing an authentication chain with a single Data Store module is displayed.

- Add the ForgeRock Authenticator (OATH) authentication module to the authentication chain as follows:
  - Click **Add a Module**.
  - 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.
  - Click **OK**.

    The graphic showing your authentication chain now includes the Data Store and ForgeRock Authenticator (OATH) authentication module.



- Save your changes.

5. Test your authentication chain as follows:
   - Log out of AM, and then go to a URL similar to the following: `https://openam.example.com:8443/openam/XUI/?`

`realm=/&service=myOATHAuthChain#login`

- Follow the procedure described in Authenticate with 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.

Authenticate with a one-time password

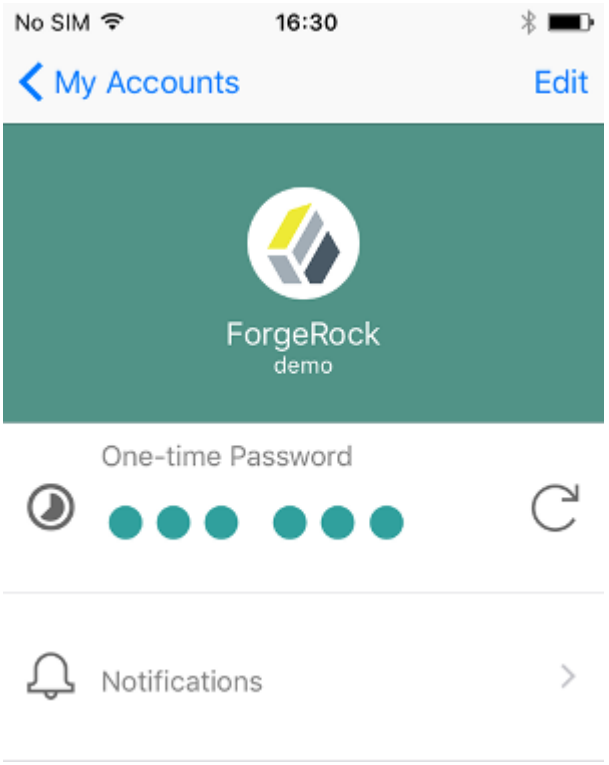
This example uses the authentication chain as created in One-time password authentication using chains.

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:

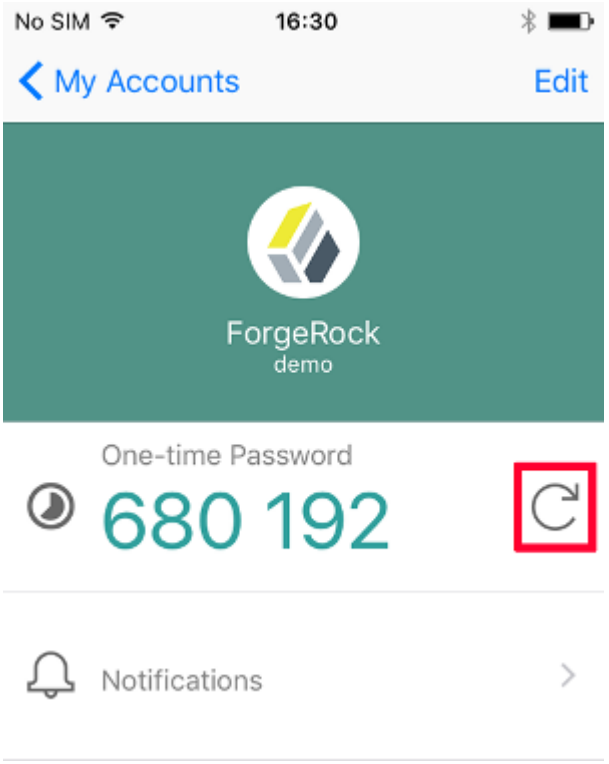
1. On your registered device, open the ForgeRock Authenticator app, and then tap the account matching the user ID you entered earlier.

   The registered authentication methods for that account are displayed:

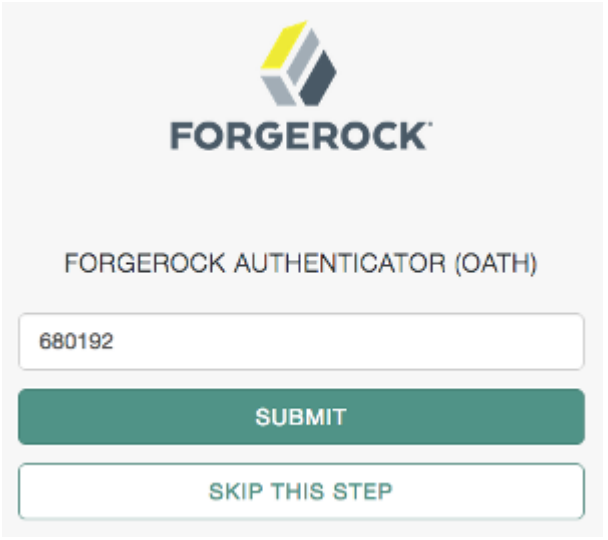


2. In the One-time Password section, click the refresh icon.

   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 click Submit:



AM will display the user's profile page.

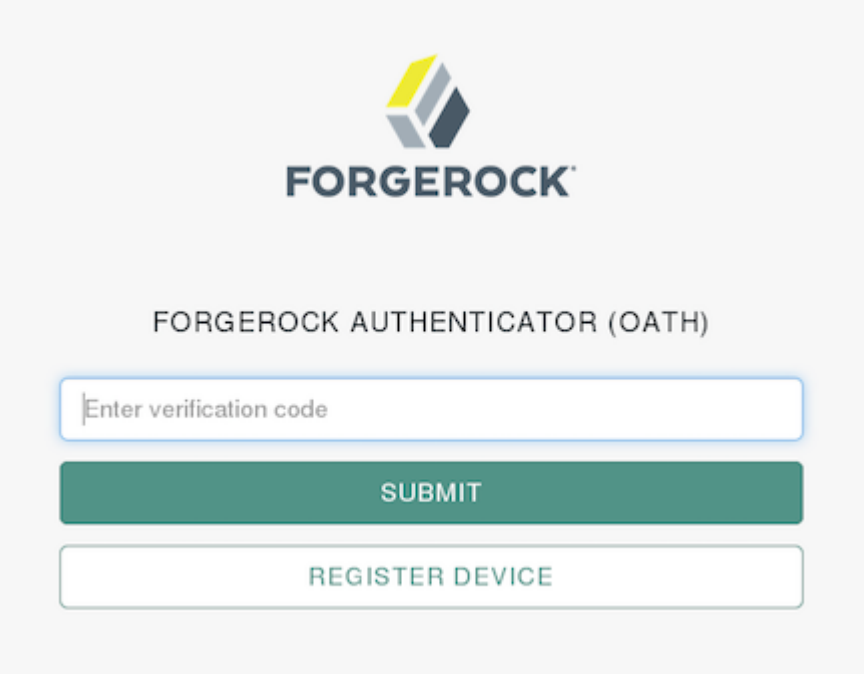
Let users opt out of one-time password authentication (OATH)

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:



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 One-time password authentication using chains. 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.

Opt out of one-time password authentication (OATH)

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. (For information about making the usage of one-time passwords mandatory in AM, see Let users opt out of one-time password authentication (OATH).) 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.

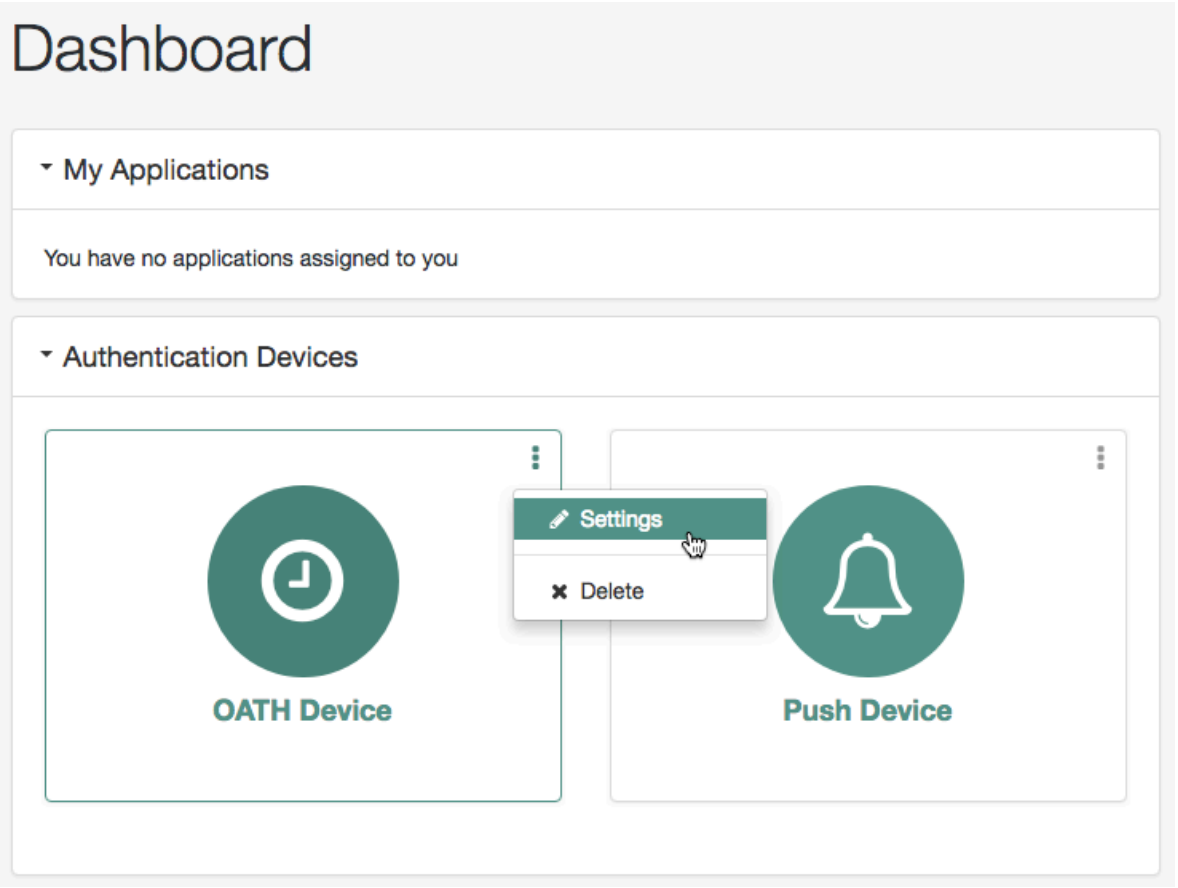
Users who decide to opt out of using one-time passwords are not prompted to enter one-time passwords when authenticating to AM.

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.

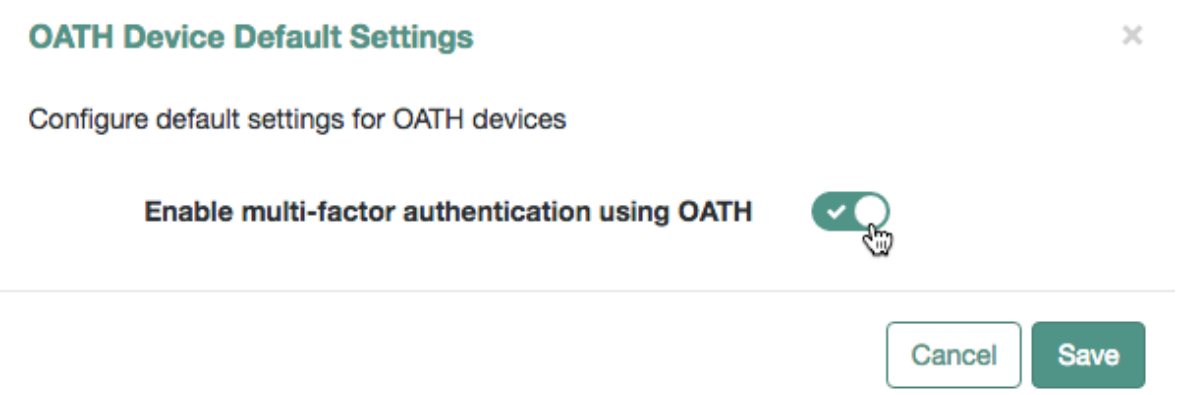
End users should follow these steps to opt out or opt in to using one-time passwords:

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 click Settings:



4. Enable or disable the multi-factor authentication option:



5. Save your work.

Manage devices for MFA

Multi-factor authentication requires you to register a device, which is used as an additional factor when you log in to AM.

The following table summarizes different tasks related to devices used for multi-factor authentication:

Task	Resources
Learn about the ForgeRock authenticator Download the ForgeRock Authenticator app, which supports push authentication notifications and one-time passwords, and register it in AM.	• The ForgeRock Authenticator app
Recover user accounts Learn how to recover a user account when the user has lost their registered device, or when their device has become out of sync with AM.	• Recover after replacing a lost device • Recover after a device becomes out of sync
Reset registered devices In some scenarios, for example, when users are not able to access their recovery codes, you may need to reset their registered devices to allow them to register again.	• Reset registered devices over REST

The ForgeRock Authenticator app

The ForgeRock Authenticator app supports push authentication notifications and one-time passwords.

Download and install the ForgeRock Authenticator app on your phone, so that you can perform multi-factor authentication. The app is available for both Android and iOS devices, and is free to download from:

- Google Play
- Apple App Store

TIP

Tip

For access to the source code for sample mobile applications, see How do I access and build the sample code provided for PingAM? in the *Knowledge Base*.

Register 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.

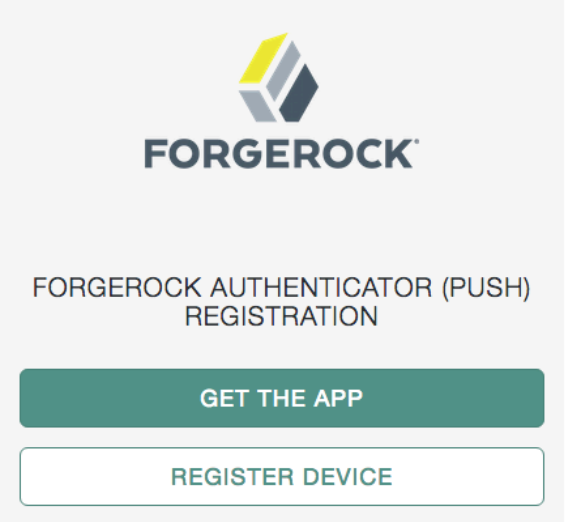
The ForgeRock Authenticator app supports registration of multiple accounts and multiple different authentication methods in each account, such as push notifications and one-time passwords.

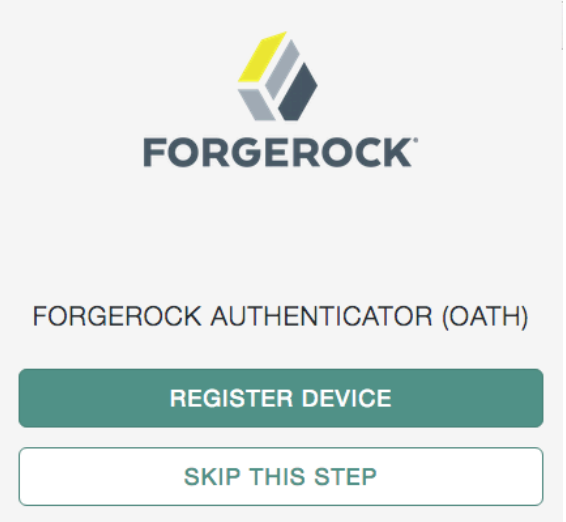
For information on registering Web Authentication (WebAuthn) devices with AM, see Create trees for Web Authentication (WebAuthn).

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.

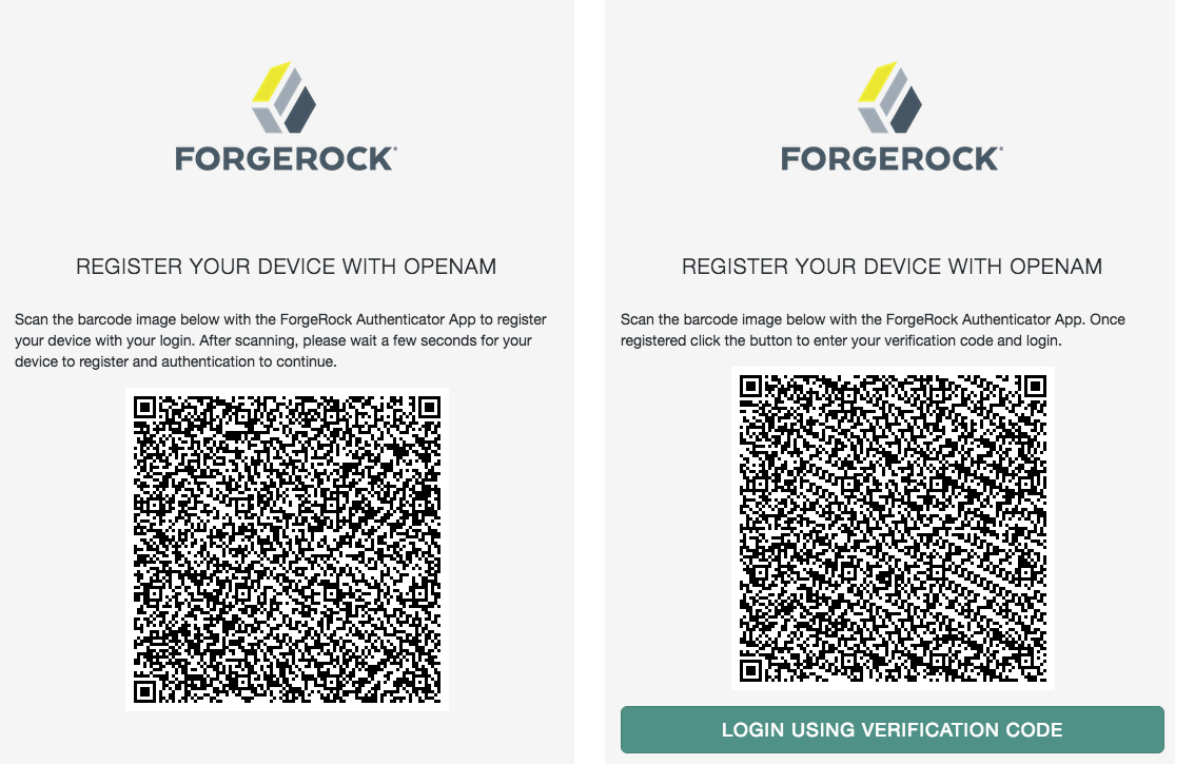
The ForgeRock Authenticator needs access to the internet to register to receive push notifications. Registering for one-time password authentication does not require a connection to the internet.

1. When visiting a protected resource without having any registered devices for multi-factor authentication, AM requires that you register a device.

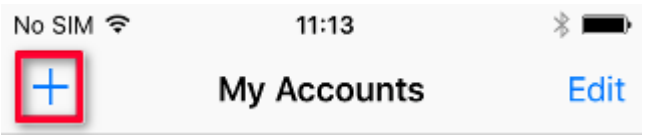




   To register your mobile phone with AM, click Register Device.



2. Start the ForgeRock Authenticator app on the device to register, and click the plus icon:

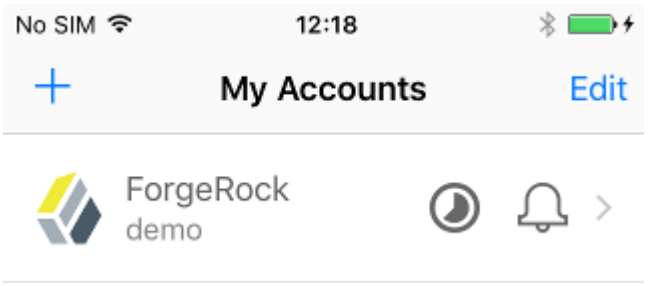


   The screen on the device changes to an interface similar to your camera app.

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 labeled **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.

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.

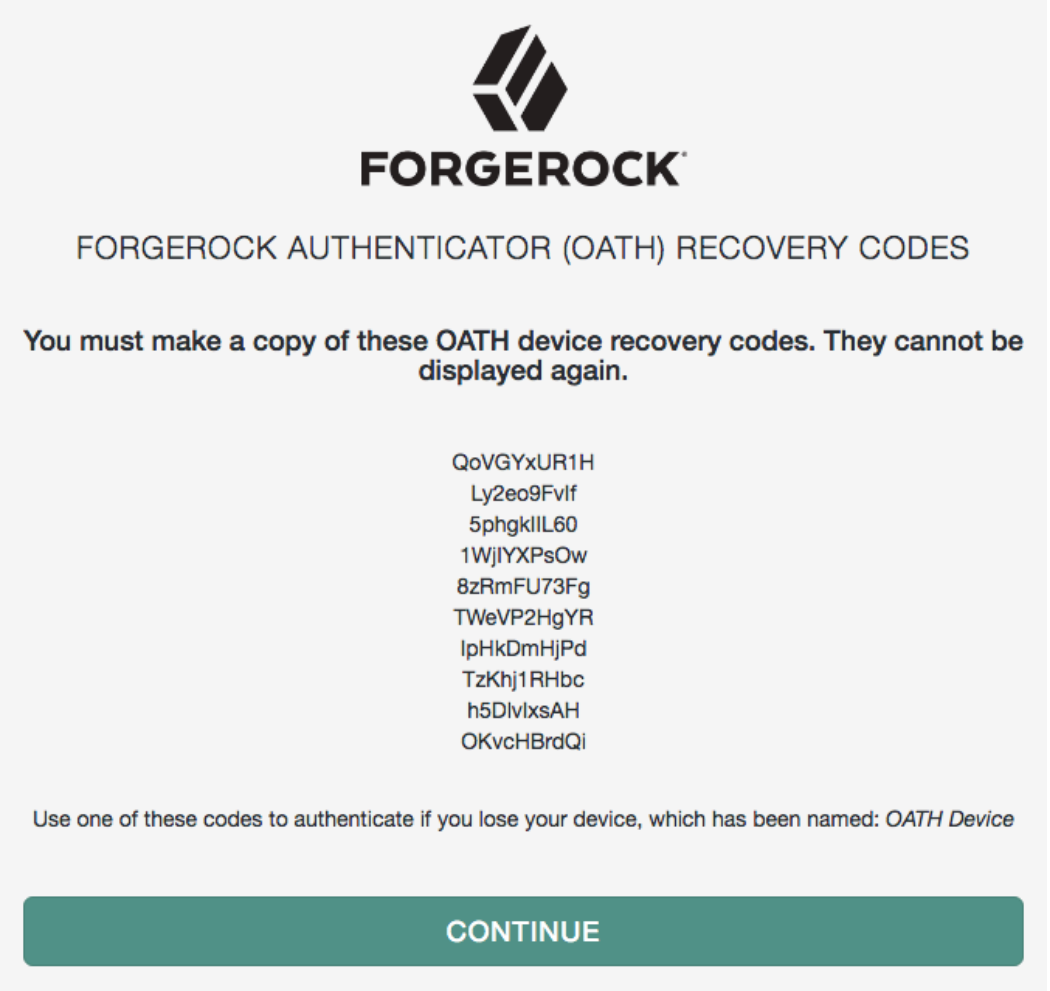
   Depending on the device type you registered, perform one of the following steps:

   a. If you registered an OATH device:

   - Click the **Login Using Verification Code** button.

     You will be asked to enter a verification code.

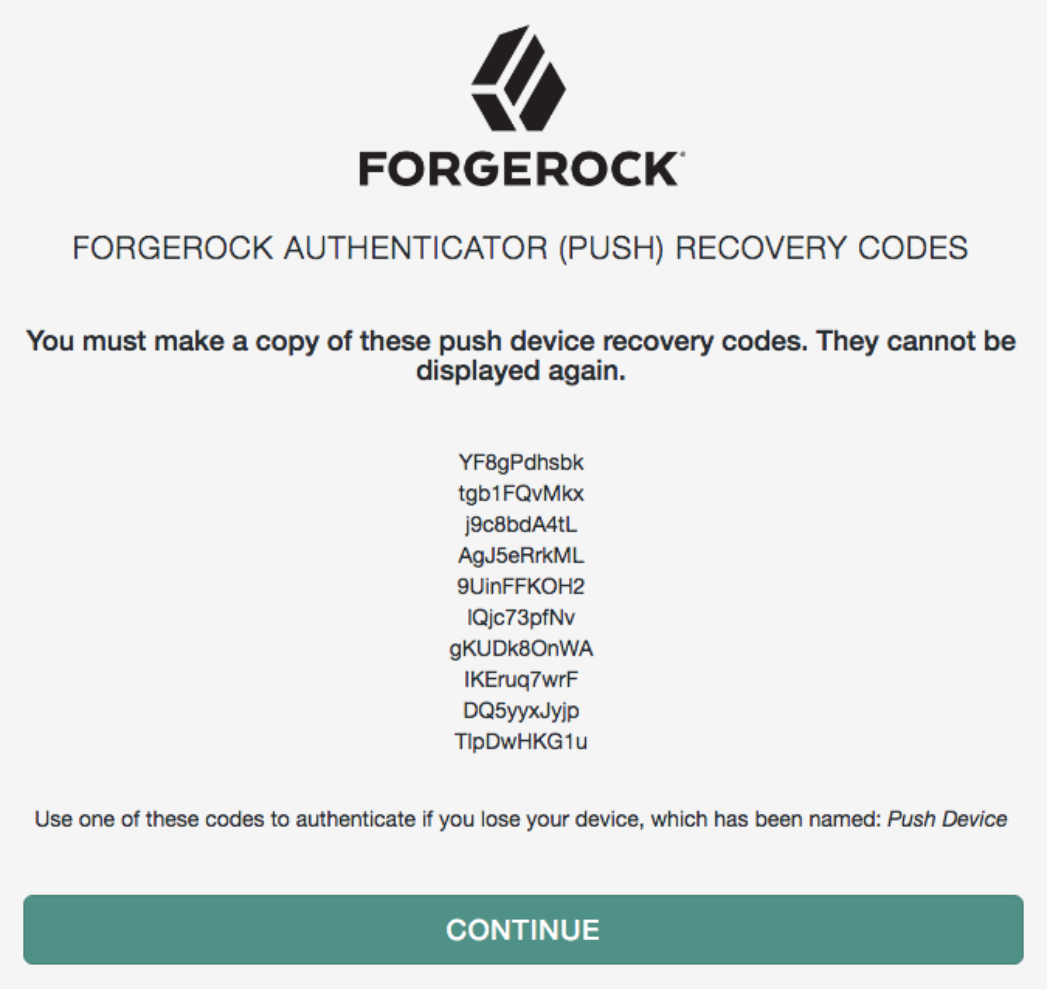
- In the ForgeRock Authenticator app, click the newly registered account, and click the refresh button to generate a new one-time password.
- Enter the one-time password into the web page, and click **Submit**.
- On the recovery codes page, make a copy of the displayed recovery codes and store them safely. The codes will never be displayed again.



When you have safely stored the recovery codes for your newly registered OATH device, click the Continue button.

b. 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.



When you have safely stored the recovery codes for your newly registered push device, click the Continue button.

Your device is registered. You can now use it to perform multi-factor authentication.

Recover after replacing a lost device

If you register a device with AM and lose it, you must authenticate to AM using a recovery code, delete the lost device, and register the new device. Perform the following steps:

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 > 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 click **Delete**.
4. If you have not already done so, install the ForgeRock Authenticator app on your new phone.

   See 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 Reset registered devices over REST for more information.

Recover 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 Recover after replacing a lost 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 Reset registered devices over REST for more information.

Reset registered devices over REST

As described in Recover 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 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.

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`.

Reset OATH devices

To reset a user's OATH device profile, send an HTTP POST request to the `/users/user/devices/2fa/oath?_action=reset` endpoint.

The following example resets the OATH devices of a user named `myUser` in a 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
}
```

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 push devices

To reset push devices over REST, send an HTTP POST request to the `/users/user/devices/2fa/push?_action=reset` endpoint.

The following example resets push devices for a user named `myUser` in a 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/push?_action=reset'
{
    "result":true
}
```

Delete WebAuthn devices

The `reset` action is not implemented on the `webauthn` endpoint.

You can delete a WebAuthn device over REST as follows:

1. Retrieve a list of WebAuthn devices associated with a user by querying the `users/`*`user`*`/devices/2fa/webauthn` endpoint.

   The following example retrieves a list of WebAuthn devices for a user named `myUser` in a realm called `mySubrealm`:

   ```
   $ curl \
   --request GET \
   --header "iplanetDirectoryPro: AQIC5w…2NzEz*" \
   'https://openam.example.com:8443/openam/json/realms/root/realms/mySubrealm/users/myUser/devices/2fa/webauthn?_queryFilter=true'
   {
     "result": [
       {
         "_id": "ff1db8bf-d2d7-46e1-926a-568b877f87a5",
         "_rev": "163664231",
         "deviceName": "New Security Key",
         "uuid": "ff1db8bf-d2d7-46e1-926a-568b877f87a5",
         "deviceManagementStatus": false
       }
     ],
     "resultCount": 1,
     "pagedResultsCookie": null,
     "totalPagedResultsPolicy": "NONE",
     "totalPagedResults": -1,
   ```

```
  "remainingPagedResults": -1
}
```

2. Delete the required WebAuthn device from the user by sending an HTTP DELETE request to the `users/*user*/devices/2fa/webauthn/*device-id*` endpoint, including the ID of the WebAuthn device you want to delete:

```
$ curl \
--request DELETE \
--header "Accept-API-Version: resource=1.0" \
--header "iplanetDirectoryPro: AQIC5w…2NzEz*" \
'https://openam.example.com:8443/openam/json/realms/root/realms/mySubrealm/users/myUser/devices/2fa/webauthn/ff1db8bf-d2d7-46e1-926a-568b877f87a5'
{
  "_id": "ff1db8bf-d2d7-46e1-926a-568b877f87a5",
  "_rev": "1642022518",
  "uuid": "ff1db8bf-d2d7-46e1-926a-568b877f87a5",
  "recoveryCodes": [
    ...
  ],
  "credentialId": "XGJpYNYv4AHG9sHHgxFfTw",
  "algorithm": "SHA256withECDSA",
  "deviceName": "New Security Key",
  "key": {
    "kty": "EC",
    "x": "zMDAfFwRQR_5HIGfK1iJQ0kYwtudesx_UXocoBVrWbo",
    "y": "UgRBOBlpq6QYsXlqfHDzY8XNv-5DEMajRp9_3klkhDU",
    "crv": "P-256"
  }
}
```

3. Repeat step 2 to delete additional WebAuthn devices for the user, if needed.

Reference

This reference covers settings and the scripting API for authentication in AM.

For global services and global authentication reference, see Global services configuration.

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, go to **Configure** > **Authentication** in the AM admin UI, and click **Core Attributes**.

To override the global core authentication configuration in a realm, go to **Realms** > ***Realm Name*** > **Authentication** > **Settings** in the AM admin UI.

`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.

NOTE

This setting applies only to modules within authentication chains.

`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 or tree used when an administrative user, such as `amAdmin`, logs in to the AM admin UI.

`ssoadm` attribute: `iplanet-am-auth-admin-auth-module`

Organization Authentication Configuration

Specifies the default authentication chain or tree 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.

NOTE

This functionality is deprecated.

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 admin UI.

```
$ ssoadm get-realm-svc-attrs \
--adminid uid=amAdmin,ou=People,dc=openam,dc=forgerock,dc=org \
--password-file PATH_TO_PWDFILE \
--realm REALM \
--servicename iPlanetAMAuthService$ ssoadm set-realm-svc-attrs \
 --adminid uid=amAdmin,ou=People,dc=openam,dc=forgerock,dc=org \
 --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 duration lockout and disables persistent (physical) lockout. *Duration lockout* means the user's account is locked 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

For duration lockout, this attribute defines a multiplier that is applied to the value of the Login Failure Lockout Duration 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 admin UI 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 admin UI 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 admin UI, 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 server-side or client-side 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.

NOTE

This setting applies only to modules within authentication chains.

amster attribute: `identityType`

ssoadm attribute: `sunAMIdentityType`

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`.

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-Side Sessions

When enabled, AM assigns *client-side* sessions to users authenticating to this realm. Otherwise, AM users authenticating to this realm are assigned *server-side* sessions.

For more information about sessions, see Introduction to sessions.

`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.

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`

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.

External Login Page URL

Specifies the URL of the external login user interface, if the authentication user interface is hosted separately from AM.

When set, AM will use the provided URL as the base of the resume URI, rather than using the Base URL Source Service to obtain the base URL. AM will use this URL when constructing the resume URI if authentication is suspended in an authentication tree.

For more information about the Base URL Source Service, see Configure the Base URL source service.

NOTE

This setting only applies to nodes within authentication trees, and does not affect modules within authentication chains.

`amster` attribute: `externalLoginPageUrl`

`ssoadm` attribute: `externalLoginPageUrl`

Default Authentication Level

Specifies the default authentication level for authentication modules.

NOTE

This setting applies only to modules within authentication chains.

`amster` attribute: `defaultAuthLevel`

`ssoadm` attribute: `iplanet-am-auth-default-auth-level`

▼ Trees

The following properties are available under the **Trees** tab, and apply only to authentication nodes and trees:

Authentication session state management scheme

Specifies the location where AM stores authentication sessions.

Possible values are:

- `CTS` . AM stores authentication sessions server-side, 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 Introduction to sessions.

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 maximum allowed duration of an authentication session, including any time spent in the suspended state, in minutes.

Values from `1` to `2147483647` are allowed.

Default: `5`

`amster` attribute: `authenticationSessionsMaxDuration`

`ssoadm` attribute: `openam-auth-authentication-sessions-max-duration`

Suspended authentication duration (minutes)

Specifies the length of time an authentication session can be suspended, in minutes.

Suspending an authentication session allows time for out-of-band authentication methods, such as responding to emailed codes or performing an action on an additional device. The value must be less than or equal to the total time allowed for an authentication session, specified in the `Max duration (minutes)` property.

Values from `1` to `2147483647` are allowed.

Default: `5`

`ssoadm` attribute: `suspendedAuthenticationTimeout`

Enable Allowlisting

When enabled, AM allowlists authentication sessions to protect them against replay attacks.

Default: Disabled

`amster` attribute: `authenticationSessionsWhitelist`

`ssoadm` attribute: `openam-auth-authentication-sessions-whitelist`

Stops sending tokenId

When HttpOnly session cookies are enabled and a client calls the `/json/authenticate` endpoint with a valid SSO token, AM returns an empty `tokenId` field.

Disable this property to have AM send a valid token ID in this scenario.

CAUTION

For security reasons, you should leave this property enabled. If you have migrated an existing deployment, adjust your clients to expect an empty token ID, then enable this property.

Default: Enabled

`amster` attribute: `authenticationTreeCookieHttpOnly`

`ssoadm` attribute: `authenticationTreeCookieHttpOnly`

▼ 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.

NOTE

This setting applies only to modules within authentication chains.

`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`

NOTE

This setting only applies to nodes within authentication trees, and does not affect modules within authentication chains.

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)

NOTE

This setting applies only to modules within authentication chains.

amster attribute: `zeroPageLoginEnabled`

ssoadm attribute: `openam.auth.zero.page.login.enabled`

Zero Page Login Referer Allowlist

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.

NOTE

This setting applies only to modules within authentication chains.

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 UI, 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 Create Post-Authentication Plugins for Chains.

NOTE

This setting applies only to modules within authentication chains.

For information about post-authentication processing for trees, see Create post-authentication hooks for trees.

`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.

NOTE

This setting applies only to modules within authentication chains.

`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`.

NOTE

This setting applies only to modules within authentication chains.

`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-side 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-side session cookies, see Session cookies and session security.

NOTE

The `User Attribute Mapping to Session Attribute` property only applies to modules within authentication chains.

For authentication trees, use the Scripted Decision node to retrieve user attributes and session properties, or the Set Session Properties node for session properties only.

`amster` attribute: `userAttributeSessionMapping`

`ssoadm` attribute: `sunAMUserAttributesSessionMapping`

Supported callbacks

For more information about how to use callbacks to authenticate to AM, see Returning callback information to AM.

The following types of callbacks are available:

Callback type	Description
Interactive	AM returns these callbacks to request information from the user.
Read-only	AM uses these callbacks to return information to the client or to show information to the user.
Backchannel	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.

Interactive callbacks

AM returns the following callbacks to request information from the user:

BooleanAttributeInputCallback

On a ForgeRock Identity Platform deployment, this callback is used to ask for a boolean-style confirmation, such as yes/no, or true/false, and retrieve the response.

Used to ask for a boolean-style confirmation, such as yes/no, or true/false, and retrieve the response.

Differs from the ConfirmationCallback in that the `BooleanAttributeInputCallback` can be used with IDM policy information to validate the input against the managed user schema. For use examples, see the Attribute Collector node.

▼ *Callback output object reference*

- `name` . A string containing the name of the attribute in the user profile.
- `prompt` . A string containing the description of the attribute. In other words, a description of the information required from the user.
- `required` . A boolean indicating whether input is required for this attribute.
- `policies` . One or more JSON objects describing validation policies that the provided input is required to pass. The object will be empty if validation is disabled in the Attribute Collector node.

  The node collects policy information from IDM. For more information about the policies available by default, see Default Policy for Managed Objects in the IDM documentation.

- `failedPolicies` . One or more JSON objects describing validation policies that the input failed. The object will only be populated after input has been submitted and validation failed.

  Requires validation to be enabled in the Attribute Collector node.

- `validateOnly` . A boolean indicating the state of this flag when previously submitted. If the UI returns this property as `true` in the input of the callback, the node will only perform input validation. The authentication journey will not continue to the next node.

  This is useful for UIs to make validation checks as the user types instead of validating the input once and continuing the journey to the next node.

  Requires validation to be enabled in the Attribute Collector node.

- `value` . A string containing a default value for the attribute, if required.

▼ *Example*


```
{
  "callbacks": [
    {
      "type": "BooleanAttributeInputCallback",
      "output": [
        {
          "name": "name",
          "value": "preferences/marketing"
        },
        {
          "name": "prompt",
          "value": "Send me special offers and services"
        },
        {
          "name": "required",
          "value": true
        },
        {
          "name": "policies",
          "value": {}
        },
        {
          "name": "failedPolicies",
          "value": []
        },
        {
          "name": "validateOnly",
          "value": false
```

```
        },
        {
          "name": "value",
          "value": false
        }
      ],
      "input": [
        {
          "name": "IDToken1",
          "value": false
        },
        {
          "name": "IDToken1validateOnly",
          "value": false
        }
      ]
    }
  ]
}
```

Return the value in the callback and the boolean that specifies whether `validateOnly` should be true.

Class to import: `org.forgerock.openam.authentication.callbacks.BooleanAttributeInputCallback`

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.

▼ 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.

▼ 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":"IDToken1",
            "value":0
         }
      ]
   }
]
```

Class to import: `javax.security.auth.callback.ConfirmationCallback`

ConsentMappingCallback

On a ForgeRock Identity Platform deployment, this callback displays managed user attributes that require user consent. It also collects consent from the user.

Used to display managed user attributes that require consent, and to collect consent from the user.

For more information about requiring consent for attributes, see Configure Privacy and Consent in the IDM documentation.

▼ Example

```
{
  "callbacks": [
    {
      "type": "ConsentMappingCallback",
      "output": [
        {
          "name": "name",
          "value": "managedUser_managedUser"
        },
        {
          "name": "displayName",
          "value": "Test Mapping"
```

```
        },
        {
          "name": "icon",
          "value": ""
        },
        {
          "name": "accessLevel",
          "value": "Actual Profile"
        },
        {
          "name": "isRequired",
          "value": true
        },
        {
          "name": "message",
          "value": "You consent to your data being shared
with external services."
        },
        {
          "name": "fields",
          "value": []
        }
      ],
      "input": [
        {
          "name": "IDToken1",
          "value": false
        }
      ]
    }
  ]
}
```

The user must give consent to all attributes, or to none. Therefore, the input object for this callback is a single boolean value.

Class to import:
`org.forgerock.openam.authentication.callbacks.ConsentMappingCallback`

DeviceProfileCallback

Used to request information about the device being used to authenticate.

The callback may request `metadata` and/or `location` information about the device by setting the relevant value to `true` in the JSON:

▼ Example

```
"callbacks": [
    {
        "type": "DeviceProfileCallback",
        "output": [
            {
                "name": "metadata",
                "value": true
            },
            {
                "name": "location",
                "value": true
            },
            {
                "name": "message",
                "value": "Collecting....."
            }
        ],
        "input": [
            {
                "name": "IDToken1",
                "value": ""
            }
        ]
    }
]
```

The callback also contains the `message` entry, with optional text to display to the user while collecting the information.

The ForgeRock SDKs gather and return the requested information in a JSON abject, as well as the following elements:

identifier
: A unique identifier string that can be used to later match the device.

alias
: A friendly name for the device, often derived from the make and model.

Return an escaped JSON in the input object on the callback. It should include information resembling the following:

▼ *Callback Response Example*

```
{
   "identifier":"aec3fe784...o3Xjiizyb9=",
   "alias":"Pixel 3 XL",
```

```
   "metadata":{
      "platform":{
         "platform":"Android",
         "version":28,
         "device":"generic_x86_arm",
         "deviceName":"AOSP on IA Emulator",
         "model":"AOSP on IA Emulator",
         "brand":"google",
         "locale":"en_US",
         "timeZone":"America/Vancouver",
         "jailBreakScore":1
      },
      "hardware":{
         "hardware":"ranchu",
         "manufacturer":"Google",
         "storage":774,
         "memory":1494,
         "cpu":4,
         "display":{
            "width":1440,
            "height":2621,
            "orientation":1
         },
         "camera":{
            "numberOfCameras":2
         }
      },
      "browser":{
         "agent":"Dalvik/2.1.0 (Linux; U; Android 9; AOSP on
IA Emulator Build/PSR1.180720.117)"
      },
      "bluetooth":{
         "supported":false
      },
      "network":{
         "connected":true
      },
      "telephony":{
         "networkCountryIso":"us",
         "carrierName":"Android"
      }
   },
   "location":{
      "latitude":51.431534,
      "Longitude":-2.622353
```

```
    }
  }
```

Class to import:
`org.forgerock.openam.authentication.callbacks.DeviceProfileCallback`

HiddenValueCallback

Used to return form values that are not visually rendered to the end user.

▼ 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`

IdPCallback

Provides the information required by a client to authenticate with a social identity provider. Clients, such as an app using the ForgeRock SDK for Android or iOS, can use this information to authenticate to the social identity provider using native APIs and SDKs provided by the mobile OS.

The Social Provider Handler node returns this callback when its **Client Type** is set to `NATIVE`.

The response to this callback should be the result of authenticating with the social provider. For example, it might be an OAuth 2.0 or OpenID Connect access or ID token, depending on the provider.

▼ Example


```
"callbacks": [
    {
        "type": "IdPCallback",
        "output": [
            {
                "name": "provider",
                "value": "amazon"
            },
            {
                "name": "clientId",
                "value": "amzn1.application-oa2-
client.f0c11aa1f8504f8da26a346ccc55a39e"
            },
            {
                "name": "redirectUri",
                "value": "https://localhost:8443/openam"
            },
            {
                "name": "scopes",
                "value": [
                    "profile"
                ]
            },
            {
                "name": "nonce",
                "value": ""
            },
            {
                "name": "acrValues",
                "value": []
            },
            {
                "name": "request",
                "value": ""
            },
            {
                "name": "requestUri",
                "value": ""
            }
        ],
```

```
        "input": [
            {
                "name": "IDToken1token",
                "value": ""
            },
            {
                "name": "IDToken1token_type",
                "value": ""
            }
        ]
    }
]
```

Class to import:
`org.forgerock.openam.authentication.callbacks.IdPCallback`

KbaCreateCallback

On a ForgeRock Identity Platform deployment, this callback collects knowledge-based authentication (KBA) answers to questions predefined in IDM, or lets users register both questions and answers.

Used to collect knowledge-based authentication (KBA) answers to predefined questions, or to let users register both questions and answers.

For more information about the predefined questions, see Configure Security Questions in the IDM documentation.

▼ Example

```
{
  "callbacks": [
    {
      "type": "KbaCreateCallback",
      "output": [
        {
          "name": "prompt",
          "value": "Select a security question"
        },
        {
          "name": "predefinedQuestions",
          "value": [
            "What's your favorite color?"
          ]
        }
      ],
      "input": [
```

```
        {
          "name": "IDToken1question",
          "value": ""
        },
        {
          "name": "IDToken1answer",
          "value": ""
        }
      ]
    }
  ]
}
```

Input objects enumerate pairs of questions and answers. When `IDToken_number_question` is empty, the value returned in `IDToken_number_answer` is related to the predefined questions. In other words, the answer collected in `IDToken1answer` is related to the first predefined question, unless `IDToken1question` is collected in as well.

Class to import:
`org.forgerock.openam.authentication.callbacks.KbaCreateCallback`

NameCallback

Used to retrieve a data string which can be entered by the user. Usually used for collecting user names.

▼ Example

```
"callbacks":[
   {
      "type":"NameCallback",
      "output":[
         {
            "name":"prompt",
            "value":"User Name"
         }
      ],
      "input":[
         {
            "name":"IDToken1",
            "value":""
         }
      ]
   }
]
```

Class to import: `javax.security.auth.callback.NameCallback`

NumberAttributeInputCallback

On a ForgeRock Identity Platform deployment, this callback collects numerical-only attributes, such as size, or age.

Used to collect numerical-only attributes, such as size, or age.

It can be used with IDM policy information to validate the input against the managed user schema. For use examples, see the Attribute Collector node.

▼ *Callback output object reference*

- `name` . A string containing the name of the attribute in the user profile.
- `prompt` . A string containing the description of the attribute. In other words, a description of the information required from the user.
- `required` . A boolean indicating whether input is required for this attribute.
- `policies` . One or more JSON objects describing validation policies that the provided input is required to pass. The object will be empty if validation is disabled in the Attribute Collector node.

  The node collects policy information from IDM. For more information about the policies available by default, see Default Policy for Managed Objects in the IDM documentation.
- `failedPolicies` . One or more JSON objects describing validation policies that the input failed. The object will only be populated after input has been submitted and validation failed.

  Requires validation to be enabled in the Attribute Collector node.
- `validateOnly` . A boolean indicating the state of this flag when previously submitted. If the UI returns this property as `true` in the input of the callback, the node will only perform input validation. The authentication journey will not continue to the next node.

  This is useful for UIs to make validation checks as the user types instead of validating the input once and continuing the journey to the next node.

  Requires validation to be enabled in the Attribute Collector node.
- `value` . A string containing a default value for the attribute, if required.

Return the numeric value in the callback and the boolean that specifies whether `validateOnly` should be true.

Class to import: `org.forgerock.openam.authentication.callbacks.NumberAttributeInputCallback`

PasswordCallback

Used to retrieve a password value.

▼ Example

```
"callbacks":[
   {
      "type":"PasswordCallback",
      "output":[
         {
            "name":"prompt",
            "value":"Password"
         }
      ],
      "input":[
         {
            "name":"IDToken1",
            "value":""
         }
      ]
   }
]
```

Class to import: `javax.security.auth.callback.PasswordCallback`

SelectIdPCallback

Offers a choice of social identity provider, or local authentication.

The Select Identity Provider node returns this callback when one or more social identity providers are enabled, or a single provider is enabled as well as the **Local Authentication** option, and therefore a choice from the user is required.

The response to this callback should be the name of the provider; for example `amazon`, or `localAuthentication`, if the user wants to authenticate without using a social provider.

▼ Example

```
"callbacks": [
    {
        "type": "SelectIdPCallback",
        "output": [
            {
                "name": "providers",
                "value": [
                    {
                        "provider": "amazon",
```

```
                        "uiConfig": {
                            "buttonCustomStyle": "background:
linear-gradient(to bottom, #f7e09f 15%,#f5c646 85%);color:
black;border-color: #b48c24;",
                            "buttonImage": "",
                            "buttonClass": "fa-amazon",
                            "buttonDisplayName": "Amazon",
                            "buttonCustomStyleHover":
"background: linear-gradient(to bottom, #f6c94e 15%,#f6c94e
85%);color: black;border-color: #b48c24;",
                            "iconClass": "fa-amazon",
                            "iconFontColor": "black",
                            "iconBackground": "#f0c14b"
                        }
                    },
                    {
                        "provider": "google",
                        "uiConfig": {
                            "buttonImage": "images/g-
logo.png",
                            "buttonCustomStyle": "background-
color: #fff; color: #757575; border-color: #ddd;",
                            "buttonClass": "",
                            "buttonCustomStyleHover": "color:
#6d6d6d; background-color: #eee; border-color: #ccc;",
                            "buttonDisplayName": "Google",
                            "iconFontColor": "white",
                            "iconClass": "fa-google",
                            "iconBackground": "#4184f3"
                        }
                    },
                    {
                        "provider": "localAuthentication"
                    }
                ]
            },
            {
                "name": "value",
                "value": ""
            }
        ],
        "input": [
            {
                "name": "IDToken1",
                "value": ""
```

```
                }
            ]
        }
]
```

Class to import:
`org.forgerock.openam.authentication.callbacks.SelectIdPCallback`

StringAttributeInputCallback

On a ForgeRock Identity Platform deployment, this callback collects string attributes, such as city names, telephone numbers, and postcodes.

Used to collect string attributes, such as city names, telephone numbers, and postcodes.

Differs from the TextInputCallback in that the `StringAttributeInputCallback` can be used to validate the input against the managed user schema policies. For use examples, see the Attribute Collector node.

▼ ***Callback output object reference***

- `name` . A string containing the name of the attribute in the user profile.
- `prompt` . A string containing the description of the attribute. In other words, a description of the information required from the user.
- `required` . A boolean indicating whether input is required for this attribute.
- `policies` . One or more JSON objects describing validation policies that the provided input is required to pass. The object will be empty if validation is disabled in the Attribute Collector node.

  The node collects policy information from IDM. For more information about the policies available by default, see Default Policy for Managed Objects in the IDM documentation.
- `failedPolicies` . One or more JSON objects describing validation policies that the input failed. The object will only be populated after input has been submitted and validation failed.

  Requires validation to be enabled in the Attribute Collector node.
- `validateOnly` . A boolean indicating the state of this flag when previously submitted. If the UI returns this property as `true` in the input of the callback, the node will only perform input validation. The authentication journey will not continue to the next node.

  This is useful for UIs to make validation checks as the user types instead of validating the input once and continuing the journey to the next node.

  Requires validation to be enabled in the Attribute Collector node.

- `value` . A string containing a default value for the attribute, if required.

▼ *Example*

```
{
  "callbacks": [
    {
      "type": "StringAttributeInputCallback",
      "output": [
        {
          "name": "name",
          "value": "givenName"
        },
        {
          "name": "prompt",
          "value": "First Name"
        },
        {
          "name": "required",
          "value": true
        },
        {
          "name": "policies",
          "value": {
            "policyRequirements": [
              "REQUIRED",
              "VALID_TYPE"
            ],
            "fallbackPolicies": null,
            "name": "givenName",
            "policies": [
              {
                "policyRequirements": [
                  "REQUIRED"
                ],
                "policyId": "required"
              },
              {
                "policyRequirements": [
                  "VALID_TYPE"
                ],
                "policyId": "valid-type",
                "params": {
                  "types": [
                    "string"
```

```
                ]
              }
            }
          ],
          "conditionalPolicies": null
        }
      },
      {
        "name": "failedPolicies",
        "value": []
      },
      {
        "name": "validateOnly",
        "value": false
      },
      {
        "name": "value",
        "value": ""
      }
    ],
    "input": [
      {
        "name": "IDToken1",
        "value": ""
      },
      {
        "name": "IDToken1validateOnly",
        "value": false
      }
    ]
  }
 ]
}
```

The following example shows the value of the policy object when input validation is not required:

```
...
{
"name": "policies",
"value": {}
},
...
```

Return the string value in the callback and the boolean that specifies whether `validateOnly` should be true.

Class to import:
`org.forgerock.openam.authentication.callbacks.StringAttributeInputCallback`

TermsAndConditionsCallback

On a ForgeRock Identity Platform deployment, this callback shows the company's terms and conditions, and collects the user's agreement to them.

Used to show the company's terms and conditions, and to collect the user's agreement to them.

To configure the terms and conditions text, see Terms and Conditions in the IDM documentation.

▼ Example

```
{
  "callbacks": [
    {
      "type": "TermsAndConditionsCallback",
      "output": [
        {
          "name": "version",
          "value": "0.0"
        },
        {
          "name": "terms",
          "value": "Terms and conditions text that customers
must agree to."
        },
        {
          "name": "createDate",
          "value": "2019-10-28T04:20:11.320Z"
        }
      ],
      "input": [
        {
          "name": "IDToken1",
          "value": false
        }
      ]
    }
```

```
   ]
}
```

The input object for this callback is a boolean that specifies whether the user agrees to the terms and conditions.

Class to import: `org.forgerock.openam.authentication.callbacks.TermsAndConditionsCallback`

TextInputCallback

Used to retrieve text input from the end user.

▼ Example

```
"callbacks":[
   {
      "type":"TextInputCallback",
      "output":[
         {
            "name":"prompt",
            "value":"Provide a nickname for this account"
         }
      ],
      "input":[
         {
            "name":"IDToken1",
            "value":""
         }
      ]
   }
]
```

Class to import: `javax.security.auth.callback.TextInputCallback`

ValidatedCreatePasswordCallback

On a ForgeRock Identity Platform deployment, this callback is used to collect a password value.

Used to collect a password value.

Differs from the PasswordCallback in that the `ValidatedCreatePasswordCallback` validates the input against the managed user schema policies. For use examples, see the Platform Password node.

▼ *Callback output object reference*

- `name` . A string containing the name of the attribute in the user profile.
- `policies` . One or more JSON objects describing validation policies that the provided input is required to pass.

  The node collects policy information from IDM. For more information about the policies available by default, see Default policy for managed objects.
- `failedPolicies` . One or more JSON objects describing validation policies that the input failed. The object will only be populated after input has been submitted and validation failed.
- `validateOnly` . A boolean indicating the state of this flag when previously submitted. If the UI returns this property as `true` in the input of the callback, the node will only perform input validation. The authentication journey will not continue to the next node.

  This is useful for UIs to make validation checks as the user types instead of validating the input once and continuing the journey to the next node.
- `prompt` . A string containing the description of the attribute. In other words, a description of the information required from the user.

▼ *Example*


```
{
  "callbacks": [
    {
      "type": "ValidatedCreatePasswordCallback",
      "output": [
        {
          "name": "echoOn",
          "value": false
        },
        {
          "name": "policies",
          "value": {
            "policyRequirements": [
              "VALID_TYPE",
              "MIN_LENGTH",
              "AT_LEAST_X_CAPITAL_LETTERS",
              "AT_LEAST_X_NUMBERS",
              "CANNOT_CONTAIN_OTHERS"
            ],
            "fallbackPolicies": null,
            "name": "password",
            "policies": [
              {
                "policyRequirements": [
```

```
        "VALID_TYPE"
      ],
      "policyId": "valid-type",
      "params": {
        "types": [
          "string"
        ]
      }
    },
    {
      "policyId": "minimum-length",
      "params": {
        "minLength": 8
      },
      "policyRequirements": [
        "MIN_LENGTH"
      ]
    },
    {
      "policyId": "at-least-X-capitals",
      "params": {
        "numCaps": 1
      },
      "policyRequirements": [
        "AT_LEAST_X_CAPITAL_LETTERS"
      ]
    },
    {
      "policyId": "at-least-X-numbers",
      "params": {
        "numNums": 1
      },
      "policyRequirements": [
        "AT_LEAST_X_NUMBERS"
      ]
    },
    {
      "policyId": "cannot-contain-others",
      "params": {
        "disallowedFields": [
          "userName",
          "givenName",
          "sn"
        ]
      },
```

```
                "policyRequirements": [
                  "CANNOT_CONTAIN_OTHERS"
                ]
              }
            ],
            "conditionalPolicies": null
          }
        },
        {
          "name": "failedPolicies",
          "value": []
        },
        {
          "name": "validateOnly",
          "value": false
        },
        {
          "name": "prompt",
          "value": "Password"
        }
      ],
      "input": [
        {
          "name": "IDToken1",
          "value": ""
        },
        {
          "name": "IDToken1validateOnly",
          "value": false
        }
      ]
    }
  ]
}
```

Return the password value in the callback and the boolean that specifies whether `validateOnly` should be true.

Class to import: `org.forgerock.openam.authentication.callbacks.ValidatedPasswordCallback`

ValidatedCreateUsernameCallback

On a ForgeRock Identity Platform deployment, this callback is used to collect user name strings.

Used to collect user name strings.

Differs from the NameCallback in that the `ValidatedCreateUsernameCallback` validates the input against the managed user schema policies. For use examples, see the Platform Username node.

▼ *Callback output object reference*

- `name` . A string containing the name of the attribute in the user profile.
- `policies` . One or more JSON objects describing validation policies that the provided input is required to pass.

  The node collects policy information from IDM. For more information about the policies available by default, see Default policy for managed objects.
- `failedPolicies` . One or more JSON objects describing validation policies that the input failed. The object will only be populated after input has been submitted and validation failed.
- `validateOnly` . A boolean indicating the state of this flag when previously submitted. If the UI returns this property as `true` in the input of the callback, the node will only perform input validation. The authentication journey will not continue to the next node.

  This is useful for UIs to make validation checks as the user types instead of validating the input once and continuing the journey to the next node.
- `prompt` . A string containing the description of the attribute. In other words, a description of the information required from the user.

▼ *Example*


```
{
  "callbacks": [
    {
      "type": "ValidatedCreateUsernameCallback",
      "output": [
        {
          "name": "policies",
          "value": {
            "policyRequirements": [
              "REQUIRED",
              "VALID_TYPE",
              "VALID_USERNAME",
              "CANNOT_CONTAIN_CHARACTERS",
              "MIN_LENGTH",
              "MAX_LENGTH"
            ],
            "fallbackPolicies": null,
```

```
"name": "userName",
"policies": [
  {
    "policyRequirements": [
      "REQUIRED"
    ],
    "policyId": "required"
  },
  {
    "policyRequirements": [
      "VALID_TYPE"
    ],
    "policyId": "valid-type",
    "params": {
      "types": [
        "string"
      ]
    }
  },
  {
    "policyId": "valid-username",
    "policyRequirements": [
      "VALID_USERNAME"
    ]
  },
  {
    "policyId": "cannot-contain-characters",
    "params": {
      "forbiddenChars": [
        "/"
      ]
    },
    "policyRequirements": [
      "CANNOT_CONTAIN_CHARACTERS"
    ]
  },
  {
    "policyId": "minimum-length",
    "params": {
      "minLength": 1
    },
    "policyRequirements": [
      "MIN_LENGTH"
    ]
  },
```

```
                {
                  "policyId": "maximum-length",
                  "params": {
                    "maxLength": 255
                  },
                  "policyRequirements": [
                    "MAX_LENGTH"
                  ]
                }
              ],
              "conditionalPolicies": null
            }
          },
          {
            "name": "failedPolicies",
            "value": []
          },
          {
            "name": "validateOnly",
            "value": false
          },
          {
            "name": "prompt",
            "value": "Username"
          }
        ],
        "input": [
          {
            "name": "IDToken1",
            "value": ""
          },
          {
            "name": "IDToken1validateOnly",
            "value": false
          }
        ]
      }
    ]
}
```

Return the user name in the callback and the boolean that specifies whether `validateOnly` should be true.

Class to import:
`org.forgerock.openam.authentication.callbacks.ValidatedUsernameCallba`

ck

Read-only Callbacks

AM uses the following callbacks to return information to the client or to show information to the user:

MetadataCallback

Used to inject key-value meta data into the authentication process. For example:

▼ 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.

▼ Example

```
"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.

For example, the Social Provider Handler node returns this callback when its **Client Type** is set to `BROWSER`, and the client needs to redirect the user to a social identity provider for authentication.

▼ Example

```
"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`

SuspendedTextOutputCallback

Used to display a message to the end user after their authentication tree is suspended.

▼ Example

```
"callbacks":[
  {
     "type": "SuspendedTextOutputCallback",
     "output": [
       {
           "name": "message",
           "value": "An email has been sent to your inbox."
       },
       {
           "name": "messageType",
           "value": "0"
       }
     ]
  }
]
```

Class to import:
`org.forgerock.openam.auth.node.api.SuspendedTextOutputCallback`

TextOutputCallback

Used to display a message to the end user.

▼ Example

```
"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`

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 module 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/realms/alpha/authenticate'
```

  The previous `curl` command URL-encodes the XML values, and the `-G` parameter appends them as query string parameters to the URL.

  NOTE

  This example applies to authentication chains only.

  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 Transactional authorization.

  - `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.

  NOTE

  This attribute applies to authentication chains only.

  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/realms/alpha/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/realms/alpha/authenticate?
  authIndexType=module&authIndexValue=DataStore'
  ```

  You should disable module-based authentication for security reasons. For more information, see Secure realms.

- `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/realms/alpha/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/realms/alpha/authenticate?
authIndexType=service&authIndexValue=ldapService'
```

If `authIndexType=service` and no `authIndexValue` is specified, the default service is used. This is similar to no `authIndexType` being set.

- `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/realms/alpha/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: Ch4ng31t" \
'https://openam.example.com:8443/openam/json/realms/root/realms
/alpha/authenticate?noSession=true'
{
    "message":"Authentication Successful",
    "successUrl":"/openam/console",
    "realm":"/"
}
```

Required: No.

Authentication nodes configuration reference

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

TIP

A number of additional authentication nodes are available from the Marketplace.

Basic authentication nodes

Use the following nodes for basic authentication tasks, such as collecting usernames and passwords:

Data Store Decision node

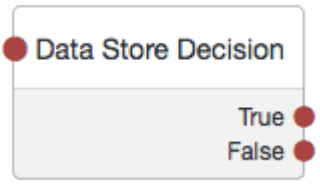
The Data Store Decision authentication node verifies that the username and password values exist in the data store configured in the realm.

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

Tree evaluation continues along the `True` path if the credentials are located in the configured data store. 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.



Properties

This node has no configurable properties.

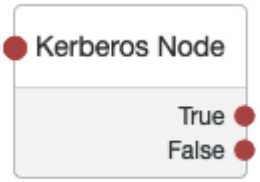
Kerberos node

The Kerberos authentication node 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.

To achieve this, the user presents a Kerberos token to AM through the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) protocol.

End users may need to set up Integrated Windows Authentication in Microsoft Edge to benefit from single sign-on when logged on to a Windows desktop.

Tree evaluation continues along the `True` path if Windows Desktop SSO is successful. Otherwise, the tree evaluation continues along the `False` path.



Properties

Property	Usage
Service Principal	Specifies the Kerberos principal for authentication in the format `HTTP/`*`AM-DOMAIN@AD-DOMAIN`*, where *AM-DOMAIN* corresponds to the host and domain names of the AM instance, and *AD-DOMAIN* is the domain name of the Kerberos realm (the FQDN of the Active Directory domain). *AD-DOMAIN* can differ from the domain name for AM. In multi-instance AM deployments, configure *AM-DOMAIN* as the FQDN or IP address of the load balancer in front of the AM instances. For example, `HTTP/AM-LB.example.com@KERBEROSREALM.INTERNAL.COM`. For more information, see the KB article How do I set up the WDSSO authentication module in PingAM in a load-balanced environment?.
Key Tab File Path	Specifies the full, absolute path of the keytab file for the specified Service Principal. TIP You generate the keytab file using the Windows **`ktpass`** utility; for example: `C:\> ktpass -out fileName.keytab -princ HTTP/openam.example.com@AD_DOMAIN.COM -pass +rdnPass -maxPass 256 -mapuser amKerberos@frdpcloud.com -crypto AES256-SHA1 -ptype KRB5_NT_PRINCIPAL -kvno 0`

Property	Usage
Kerberos Realm	Specifies the name of the Kerberos (Active Directory) realm used for authentication. Must be specified in all caps.
Kerberos Server Name	Specifies the fully qualified domain name, or IP address of the Kerberos (Active Directory) server.
Trusted Kerberos realms	Specifies a 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. Each trusted Kerberos realm must be specified in all caps.
Return Principal with Domain Name	When enabled, AM returns the fully qualified name of the authenticated user rather than just the username.
Lookup User In Realm	Validates the user against the configured data stores. If the user from the Kerberos token is not found, tree evaluation continues along the `False` path. This search uses the `Alias Search Attribute Name` from the core realm attributes. See User Profile for more information about this property.
Is Initiator	When enabled (`true`), specifies that the node is using *initiator* credentials, which is the default. When disabled (`false`), specifies that the node is using *acceptor* credentials.

Example

This flow will attempt to authenticate the user, by using Windows Desktop SSO. If unsuccessful, AM will request the username and password for login. Meter nodes are used to track metrics for the various paths through the tree.

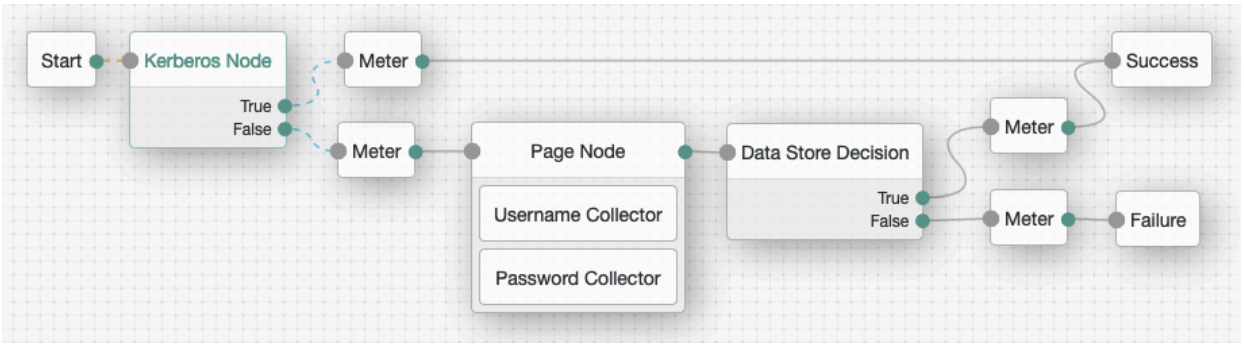


Figure 19. Kerberos Example Tree

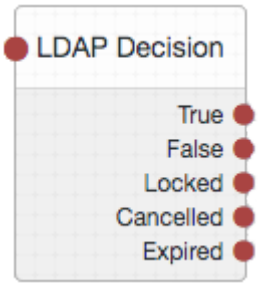
LDAP Decision node

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 node and Password Collector node, or by using the Zero Page Login Collector node.

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 user needs to change their password on first login, but cancels the password change form, tree evaluation continues along the `Cancelled` outcome path.

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

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. 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.

Property	Usage
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`. Note that if you have specified multiple attribute values, you must also add those attributes to the `Alias Search Attribute Name` property when using account lockout. See User Profile for more information about this property.
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.

Property	Usage
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.
Minimum Password Length	Specifies the minimum acceptable password length. Default: `8`

Property	Usage
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`. Set the units for the interval in the `LDAP Connection Heartbeat Time Unit` property. Note that setting this property to `0` will only ensure default values apply, and will *not* disable the heartbeat (keepalive) or load balancer availability checks. This can only be configured at the global level. 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 seconds, that AM should wait for a response from the directory server. Default: `0` (means no timeout)

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.

Tree evaluation continues along the single outcome path after capturing the password.

Properties

This node has no configurable properties.

Username Collector node

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

Tree evaluation continues along the single outcome path after capturing the username.

Properties

This node has no configurable properties.

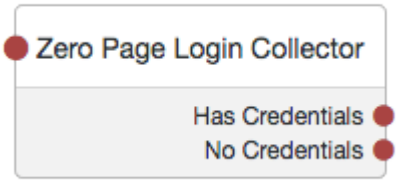
Zero Page Login Collector node

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.

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.

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

Property	Usage
Username Header name	Enter the name of the header that contains the username value. 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 allowlist property. Default: `Enabled`

Property	Usage
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 allowlist causes tree evaluation to continue along the `No Credentials` outcome path. NOTE You must disable the **Allow Without Referer** property for the referer allowlist property to take effect.

Example

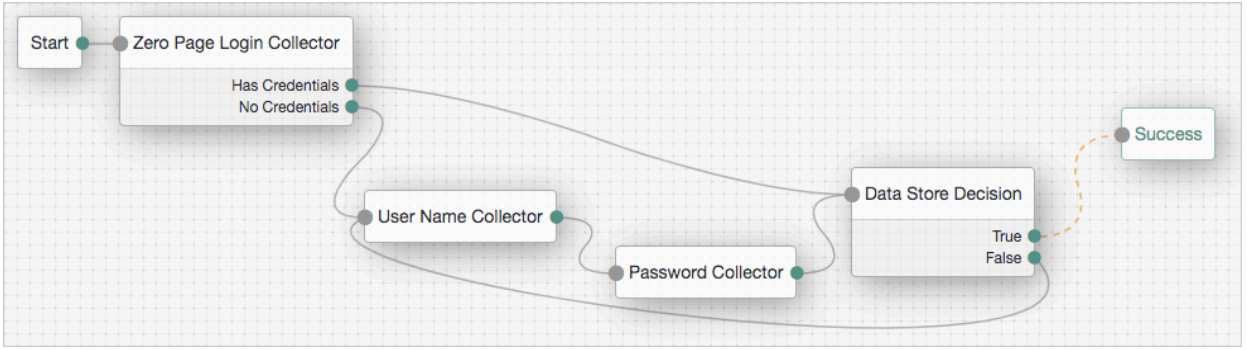


Figure 20. Example Tree With Zero Page Login node

Multi-factor authentication nodes

Use the following nodes to configure trees with multi-factor authentication capabilities, such as web authentication and push authentication:

Get Authenticator App node

The Get Authenticator App node presents the user with links to obtain your authenticator app from the Apple App Store or the Google Play store.

Tree evaluation continues along the single outcome path when the user clicks the Continue button.

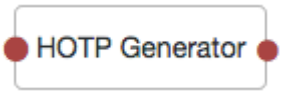
Properties

Property	Usage
Get App Authenticator Message	Optional. Localized title for the node. The key is the language (such as `en` or `fr`), and the value is the message to display.
Continue Label	Optional. Localized text to use on the Continue button. The key is the language (such as `en` or `fr`), and the value is the message to display.
Apple App Store URL	Specifies the URL to download your authenticator app from the Apple App Store. The default value points to the ForgeRock Authenticator app for iOS.
Google Play URL	Specifies the URL to download your authenticator app from the Google Play Store. The default value points to the ForgeRock Authenticator app for Android.

HOTP Generator node

The HOTP Generator authentication node creates a string of random digits, of the length specified. The default length is 8 digits.

Passwords are stored in the `oneTimePassword` transient state property of the authentication tree.



Properties

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

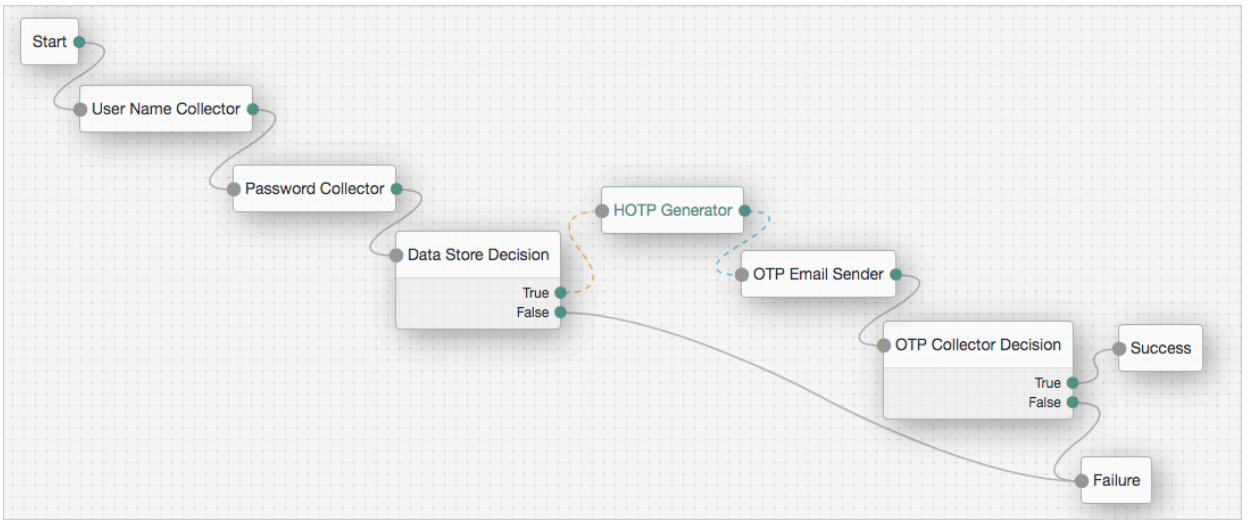


Figure 21. HmacOneTimePassword Tree With HOTP Generator node (Standalone AM)

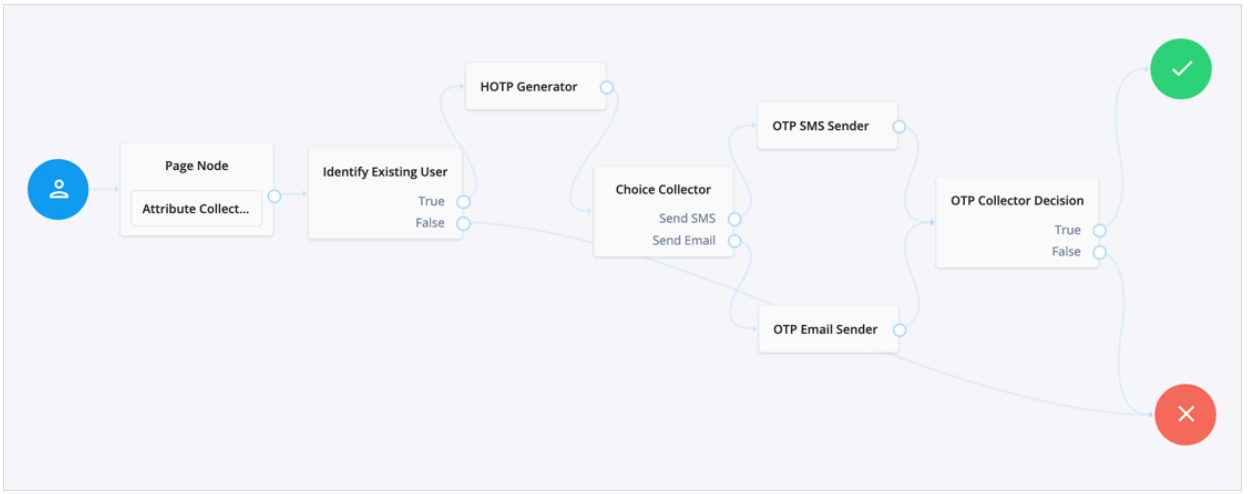


Figure 22. HmacOneTimePassword Tree With HOTP Generator node (ForgeRock Identity Platform)

MFA Registration Options node

The MFA Registration Options node lets the user register a multi-factor authentication device or skip the registration process.

The node requires the username of the identity to update; for example, by using a Username Collector node and also the type of MFA device; for example, by placing a Push Sender node earlier in the authentication journey.

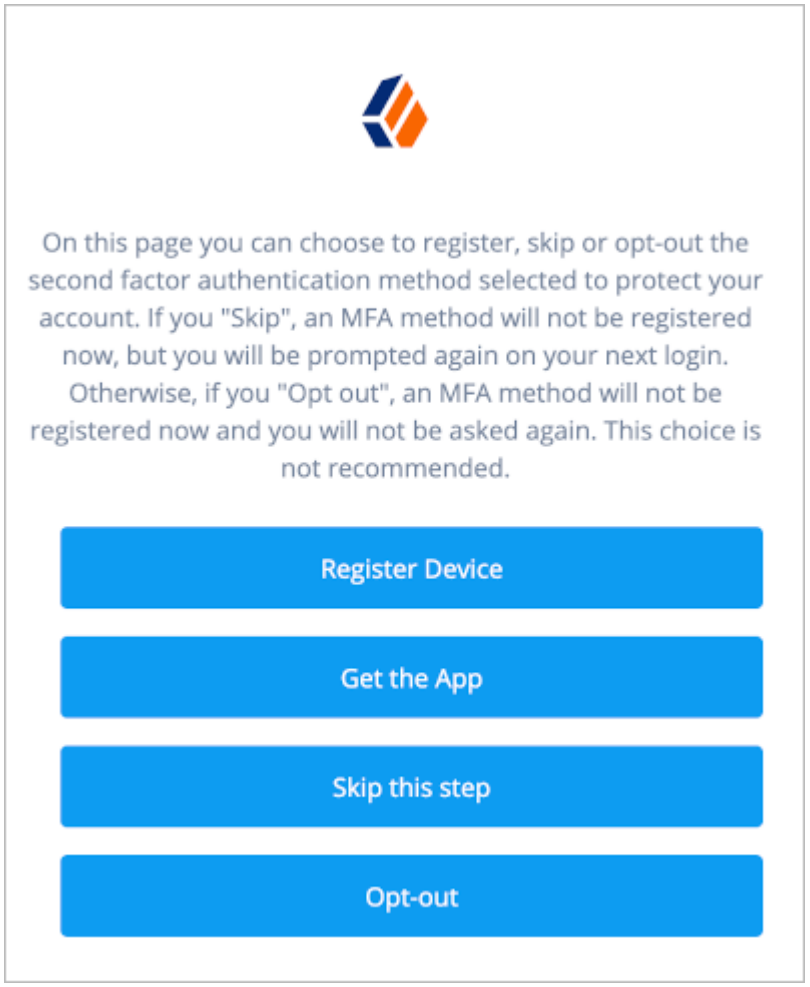
Properties

Property	Usage
Remove 'skip' option	Localized title for the node. The key is the language (such as `en` or `fr`), and the value is the message to display.
Display Get Authenticator App	Localized text to use on the Continue button. The key is the language (such as `en` or `fr`), and the value is the message to display.
Message	Localized text to use as the title of the screen. The key is the language (such as `en` or `fr`), and the value is the message to display.
Register Device	Localized text to use on the Register Device button. The key is the language (such as `en` or `fr`), and the value is the message to display.
Get Authenticator App	Localized text to use on the Get Authenticator App button. The key is the language (such as `en` or `fr`), and the value is the message to display.
Skip this Step	Localized text to use on the Skip this Step button. The button, and the outcome, only appear if the **Remove 'skip' option** is not enabled. The key is the language (such as `en` or `fr`), and the value is the message to display.

Property	Usage
Opt-out	Localized text to use on the Opt-Out button. The button, and the outcome, only appear if the **Remove 'skip' option** is not enabled. Note that the node itself does not affect the users' profile. Connect the **Opt-out** outcome to an Opt-out Multi-Factor Authentication node to actually persist the ability to skip MFA to the users' profile. The key is the language (such as `en` or `fr`), and the value is the message to display.

Tree evaluation continues along whichever outcome the user selects when presented with the options.

Example



OATH Token Verifier node

The OATH Token Verifier node requests and verifies a one-time password (OTP) generated by a device such as a mobile phone. The default configuration is time-based OTP (TOTP), but the node also supports HMAC (HOTP).

The node requires that the user credentials are authenticated, and that the user has previously registered a device using the OATH Registration node. These two nodes work together to provide all the capabilities of a secure OATH authentication journey.

They can also be used in combination with other MFA nodes to extend these capabilities, for example:

- Get Authenticator App node
- Opt-out Multi-Factor Authentication node
- MFA Registration Options node

NOTE

You can use the OATH nodes in conjunction with the ForgeRock Authenticator app to register your phone, receive notifications, or generate one-time passwords.

For a visual overview of how the OATH nodes can be used within an authentication tree layout, see the Example.

Properties

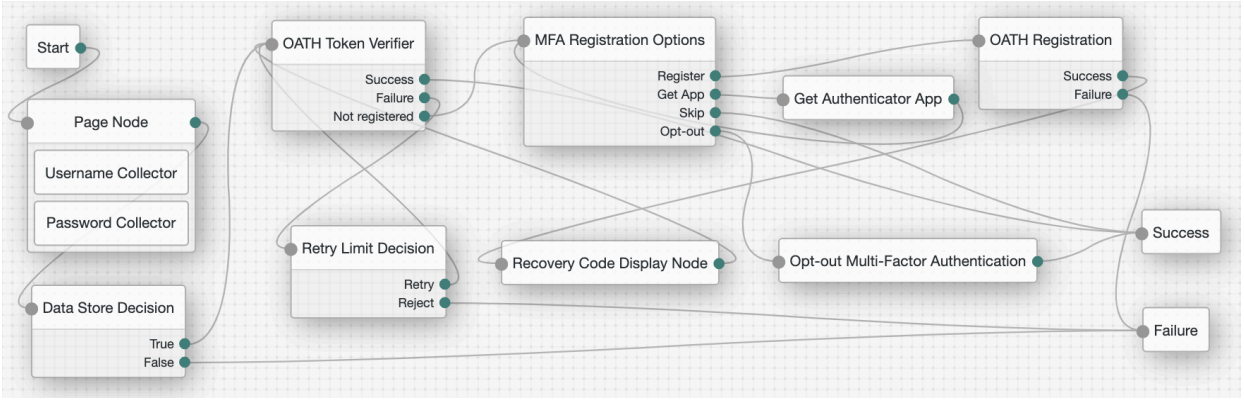
Property	Usage
OATH Algorithm	Specify the algorithm your device uses to generate the OTP: *HOTP* HOTP uses a counter value that is incremented every time a new OTP is generated. *TOTP* TOTP generates a new OTP every few seconds as specified by the `TOTP Time Step Interval` value. The default value is TOTP. If this is changed to HOTP, you need to set the same value in the OATH Registration node.
HOTP Window Size	This property sets 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, the server will accept an OTP that is generated between counter 3 and 102.
TOTP Time Step Interval	The length of time that an OTP is valid, in seconds. For example, if the time step interval is 30 seconds, a new OTP will be generated every 30 seconds, and it will be valid for 30 seconds only. The default value is 30.
TOTP Time Steps	This is the number of time step intervals that the OTP is permitted to be out of sync. This applies to codes that are generated before or after the current code. For example, with a time step of 1, the server will permit either the previous, the current, or the next code. The default value is 2.

Property	Usage
TOTP Hash Algorithm	The HMAC hash algorithm to be used to generate the OTP codes. ForgeRock Authenticator (OATH) supports SHA1, SHA256, and SHA512.
TOTP Maximum Allowed Clock Drift	Number of time steps a client is allowed to be out of sync with the server before a manual resynchronization is required. For example, with 3 allowed drifts and a time step interval of 30 seconds, the server will allow codes from up to 90 seconds from the current time to be treated as the current time step. The drift for a user's device is calculated each time they enter a new code. If the drift exceeds this value, the user's authentication code will be rejected.
Allow recovery codes	Specify whether to allow users to use one of the recovery codes to proceed with the login.

Tree evaluation continues along one of the following outcome paths:

- `Not registered` : If there is no registered device for the user.
- `Failure` : If the user is not authenticated, or the collected token code cannot be verified.
- `Success` : If there is a registered device and the token code is verified.

Example



OATH Registration node

The OATH Registration node lets the user register a device for OATH-based multi-factor authentication (MFA). Based on the node properties, the user device displays a QR code that includes all the details required for registration. If registration is successful, the node stores the device data, recovery codes (if enabled), and sets the skippable attribute to prevent repeat registration at next login.

The node requires the credentials of the user; for example, by using a sequence of the following nodes earlier in the authentication journey:

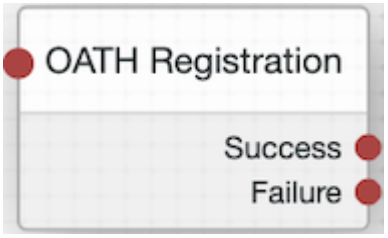
- Username Collector node
- Password Collector node
- Data Store Decision node

Connect the OATH Registration node's `Success` outcome path to the OATH Token Verifier node to continue to OTP verification.

NOTE

You can use the OATH nodes in conjunction with the ForgeRock Authenticator app to register your phone, receive notifications, or generate one-time passwords.

View the OATH Token Verifier node example to see how these nodes can be used in combination with other MFA nodes to create a complete OATH authentication journey.



Properties

Property	Usage
Issuer	Specify an identifier to appear on the user's device, such as a company name, a website, or an AM realm. The value is displayed by the authenticator app.
Account Name	Define the profile attribute to display as the username in the authenticator app. If not specified, or if the specified profile attribute is empty, their username is used.

Property	Usage
Background Color	The background color, in hex notation, to display behind the issuer's logo within the authenticator app.
Logo Image URL	The location of an image to download and display as the issuer's logo within the authenticator app.
Generate Recovery Codes	If enabled, recovery codes are generated and stored in the success outcome's transient state. Use the Recovery Code Display node to display the codes to the user for safekeeping.
One Time Password Length	The length of the generated OTP in digits. This value must be at least 6, and compatible with the hardware/software OTP generators you expect end users to use. For example, Google and ForgeRock authenticators support values of 6 and 8 respectively.
Minimum Secret Key Length	Number of hexadecimal characters allowed for the Secret Key.
OATH Algorithm	Specify the algorithm your device uses to generate the OTP: *HOTP* HOTP uses a counter value that is incremented every time a new OTP is generated. *TOTP* TOTP generates a new OTP every few seconds as specified by the `TOTP Time Step Interval` value. The default value is TOTP. If this is changed to HOTP, you need to set the same value in the OATH Token Verifier node.

Property	Usage
TOTP Time Step Interval	The length of time that an OTP is valid, in seconds. For example, if the time step interval is 30 seconds, a new OTP will be generated every 30 seconds, and it will be valid for 30 seconds only. The default value is 30.
TOTP Hash Algorithm	The HMAC hash algorithm to be used to generate the OTP codes. AM supports SHA1, SHA256, and SHA512.
HOTP Checksum Digit	This adds a digit to the end of the OTP generated to be used as a checksum to verify the OTP was generated correctly. This is in addition to the actual password length. You should only set this if your device supports it.
HOTP Truncation Offset	This is an option used by the HOTP algorithm that not all devices support. This should be left as the default value of -1, unless you know your device uses an offset.
QR code message	The message with instructions to scan the QR code to register the device. Click **Add**. Enter the message locale in the `Key` field; for example `en-gb`. Enter the message to display to the user in the `Value` field.

If registration is succesful and the device details are stored, tree evaluation continues along the `Success` outcome path. If AM encounters an issue during the registration process or the user fails to complete registration, evaluation proceeds along the `Failure` path.

Opt-out Multi-Factor Authentication node

The Opt-out Multi-Factor Authentication node sets the Skippable attribute in the user's profile, which lets them skip MFA.

The node requires the username of the identity to update; for example, by using a Username Collector node and also the type of MFA device to set as "skippable." For

example, by placing a Push Sender node node earlier in the authentication journey.

Tree evaluation continues along the single outcome path after setting the MFA device as "skippable" in the users' profile.

Properties

This node has no configurable properties.

OTP Collector Decision node

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

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

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.
- 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 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.

Properties

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.

Property	Usage
Email From Address	Specifies the email address from which the one-time password will appear to have been sent.
Email Attribute Name	Specifies the user's profile attribute containing the email address to which to email the OTP. Default: `mail`
The subject of the email	Click **Add** to add a new email subject. Enter the locale (for example, `en-uk`) in the **KEY** field and the subject in the **VALUE** field. Repeat these steps for each locale that you support.
The content of the email	Click **Add** to add the content of the email. Enter the locale (for example, `en-uk`) in the **KEY** field and the email content in the **VALUE** field. Repeat these steps for each locale that you support.
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`

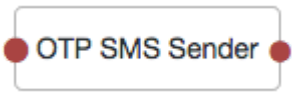
OTP SMS Sender node

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

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.
The subject of the message	Click **Add** to add a new message subject. Enter the locale (for example, `en-uk`) in the **KEY** field and the subject in the **VALUE** field. Repeat these steps for each locale that you support.
The content of the message	Click **Add** to add the content of the message. Enter the locale (for example, `en-uk`) in the **KEY** field and the email content in the **VALUE** field. Repeat these steps for each locale that you support.
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`

Push Registration node

The Push Registration authentication node provides a way to register a device, such as a mobile phone, for multi-factor authentication using push notifications. For more information, see MFA: Push authentication.

If the user successfully registers their authenticator, then tree evaluation continues along the `Success` outcome path.

If the node does not receive a response from the users' device within the time specified in the node configuration, evaluation continues along the `Time Out` outcome path.

If AM encounters an issue when attempting to register using a device, tree evaluation continues along the `Failure` outcome path.

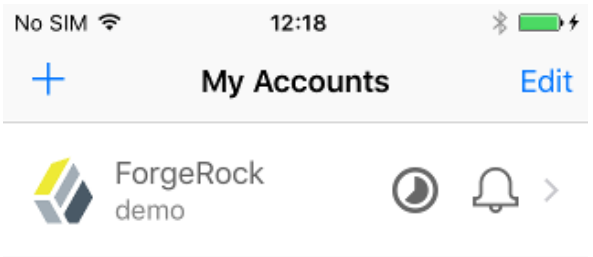
The node requires the username of the identity to update; for example, by using a Username Collector node.

You must also configure the *Push Notification Service*.

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.

Properties

Property	Usage
Issuer	Specify an identifier so that the user knows which service their account relates to. The value is displayed by the authenticator app:  For example, `Example Inc.`, or the name of your application.

Property	Usage
Account Name	Specifies the profile attribute to display as the username in the authenticator app. If not specified, or if the specified profile attribute is empty, their username is used.
Registration Response Timeout	Specify the number of seconds to wait for a response from the authenticator. If the specified time is reached, tree evaluation continues along the `Time Out` outcome path.
Background Color	Specifies the background color, in hex notation, to display behind the issuer's logo within the ForgeRock Authenticator app.
Logo Image URL	Specifies the location of an image to download and display as the issuer's logo within the ForgeRock Authenticator app.
Generate Recovery Codes	Specify whether push-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 push-specific recovery codes. Only the most recent set of recovery codes can be used for authentication if a device has been lost or stolen.

Property	Usage
QR code message	The message with instructions to scan the QR code to register the device. Click **Add**. Enter the message locale in the `Key` field; for example `en-gb`. Enter the message to display to the user in the `Value` field.

Example

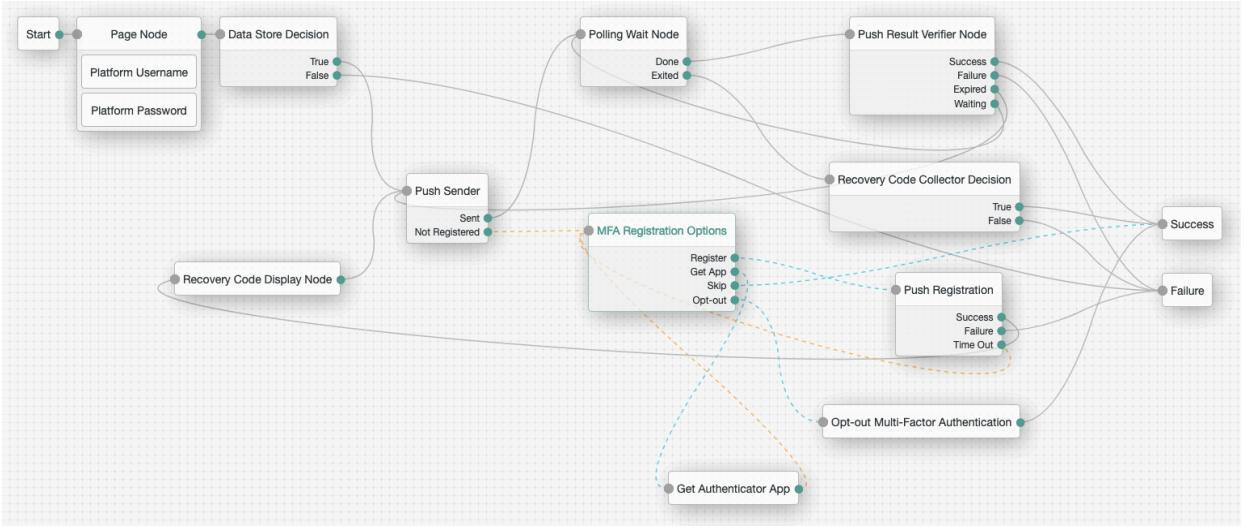


Figure 23. Example Push Tree (Standalone AM)

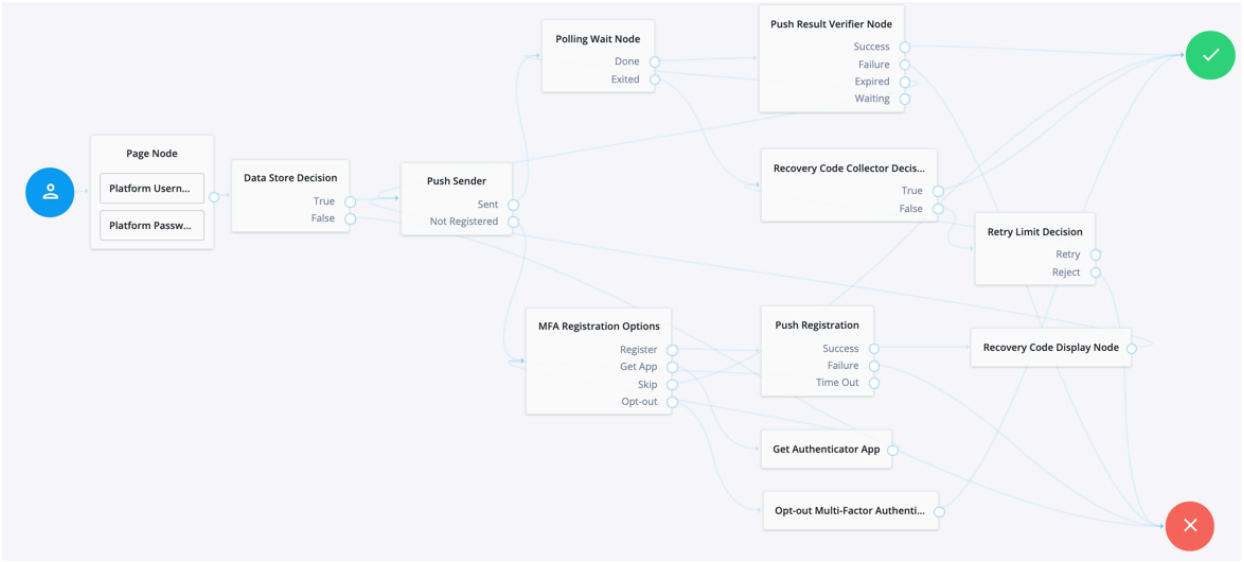
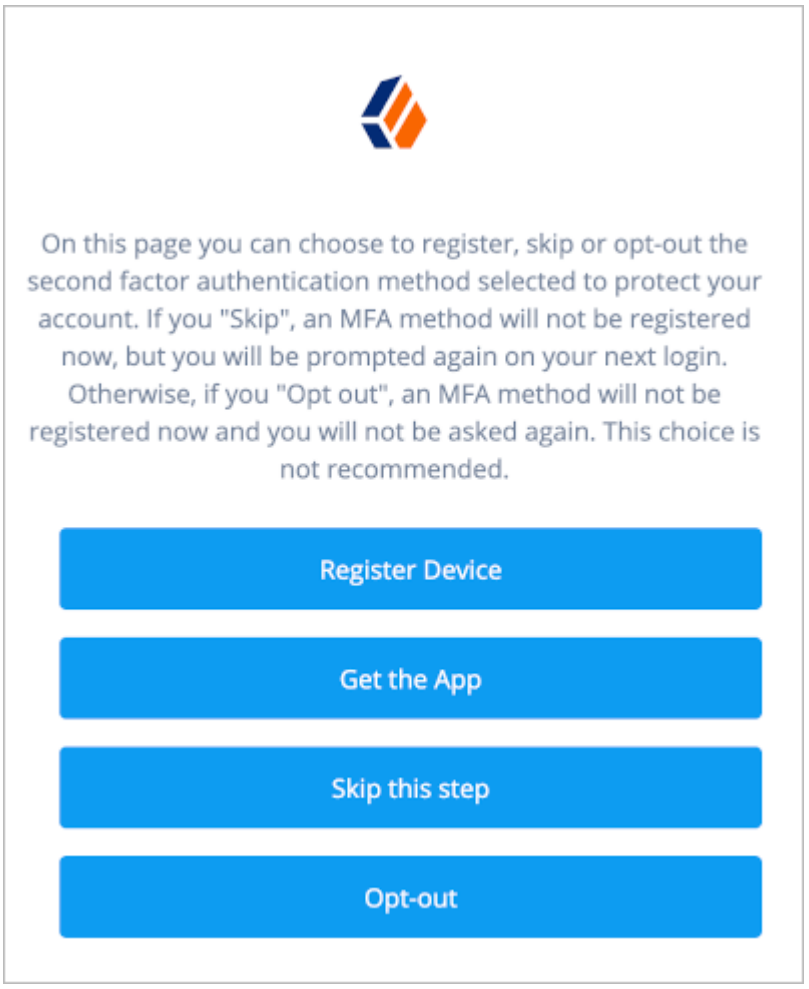


Figure 24. Example Push Tree (ForgeRock Identity Platform)

The example tree above shows a possible implementation of a tree for handling push devices.

After verifying the users credentials against the configured data store, tree evaluation continues to the Push Sender node.

If the user does not yet have a registered device, the MFA Registration Options node displays the following options:



Register Device
: The journey continues to the Push Registration node, which displays the QR code that should be scanned with a suitable authenticator app.

Get the App
: The journey continues to the Get Authenticator App node, which displays the links needed to obtain a suitable app; for example, the ForgeRock Authenticator.

Skip this step
: Displayed only if the node configuration lets the user skip. In this example tree, skipping is linked to the Success node.

Opt-out
: Displayed only if the node configuration allows the user to skip or opt out. The journey continues to the Opt-out Multi-Factor Authentication node, which updates the users' profile to skip MFA with push in the future. In this example, after updating the profile the journey continues to the Success node.

Once the registration is complete the path returns to the Push Sender node, which starts the actual push notification stage of the journey.

A polling loop using the Polling Wait node in combination with the Push Result Verifier node continuously checks whether the user has successfully responded to the push notification.

An option displayed on the Polling Wait node lets the user exit that loop, and instead provide one of their push-specific recovery codes, letting them log in if they have lost their device, for example.

Note that in order for a user to manage their registered push devices, they must log in using either the device, or a recovery code. For more information, see Manage devices for MFA.

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.

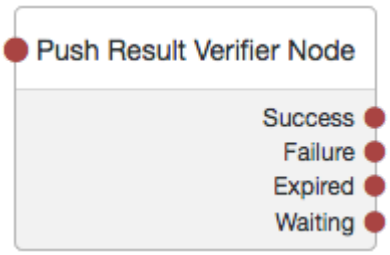
Tree evaluation continues along the `Failure` outcome path if the push notification was negatively responded to by the user. For example, using the ForgeRock Authenticator app, the user tapped the cancel icon in the top-right of the screen.

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 `nodeState` object of the tree, for the `pushContent` key. For information on creating or customizing authentication nodes, see Node development.



Properties

This node has no configurable properties.

Push Sender node

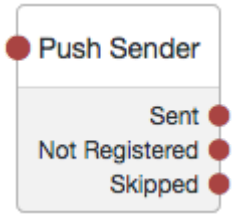
The Push Sender authentication node sends push notification messages to a device such as a mobile phone, enabling multi-factor authentication.

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. 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 or Platform Username node.

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 (OATH).



Properties

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.

Property	Usage
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. Messages provided in the node override the defaults provided by AM. For information about customizing and translating the default messages, see Internationalization. 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}}.`

Property	Usage
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

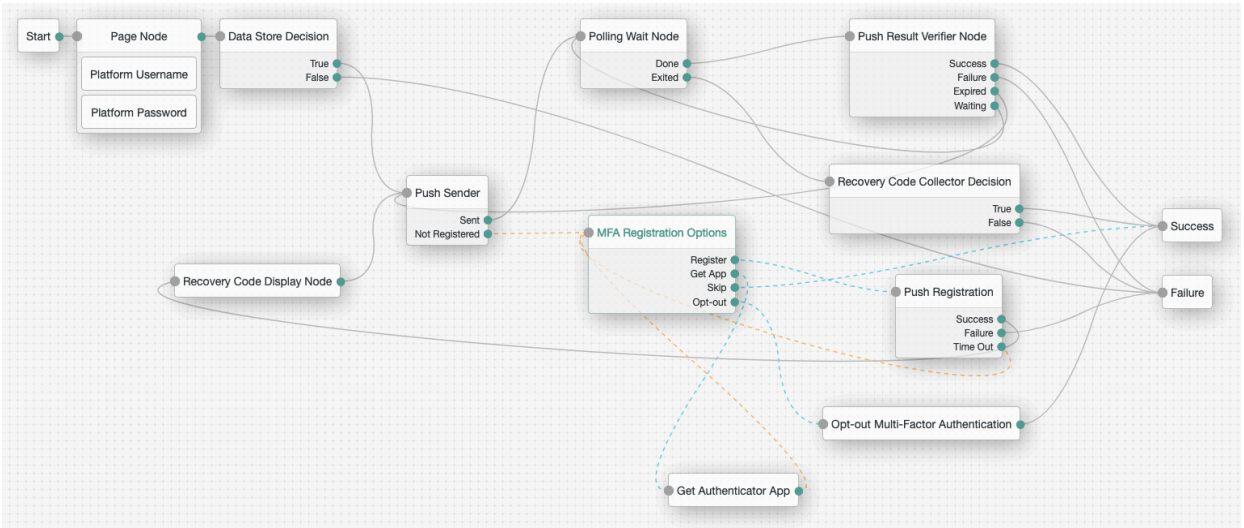


Figure 25. Example Push Tree (Standalone AM)

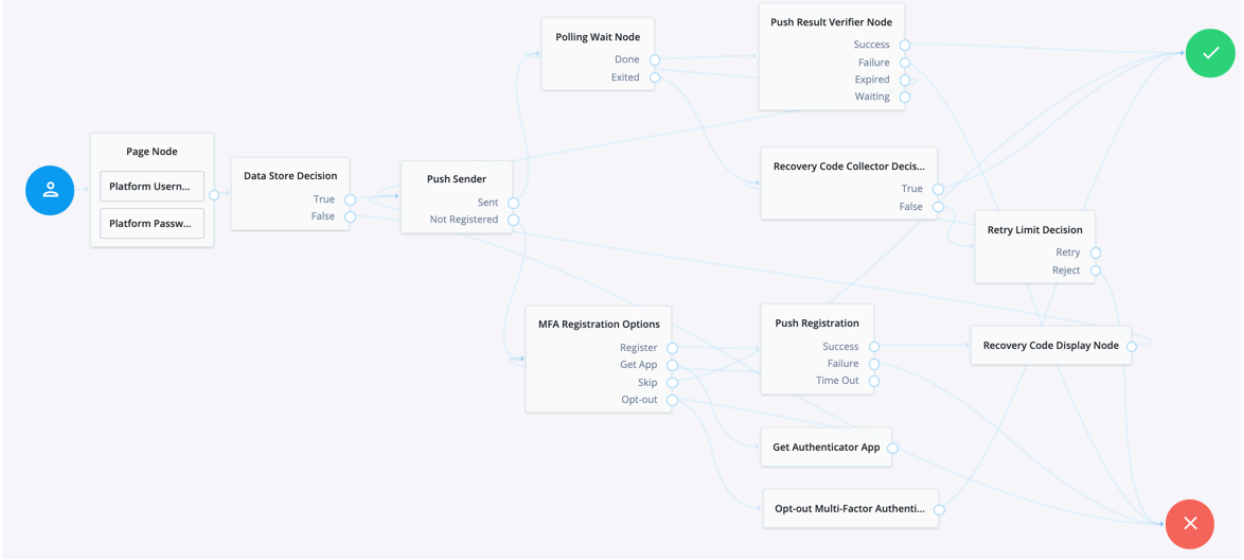


Figure 26. Example Push Tree (ForgeRock Identity Platform)

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

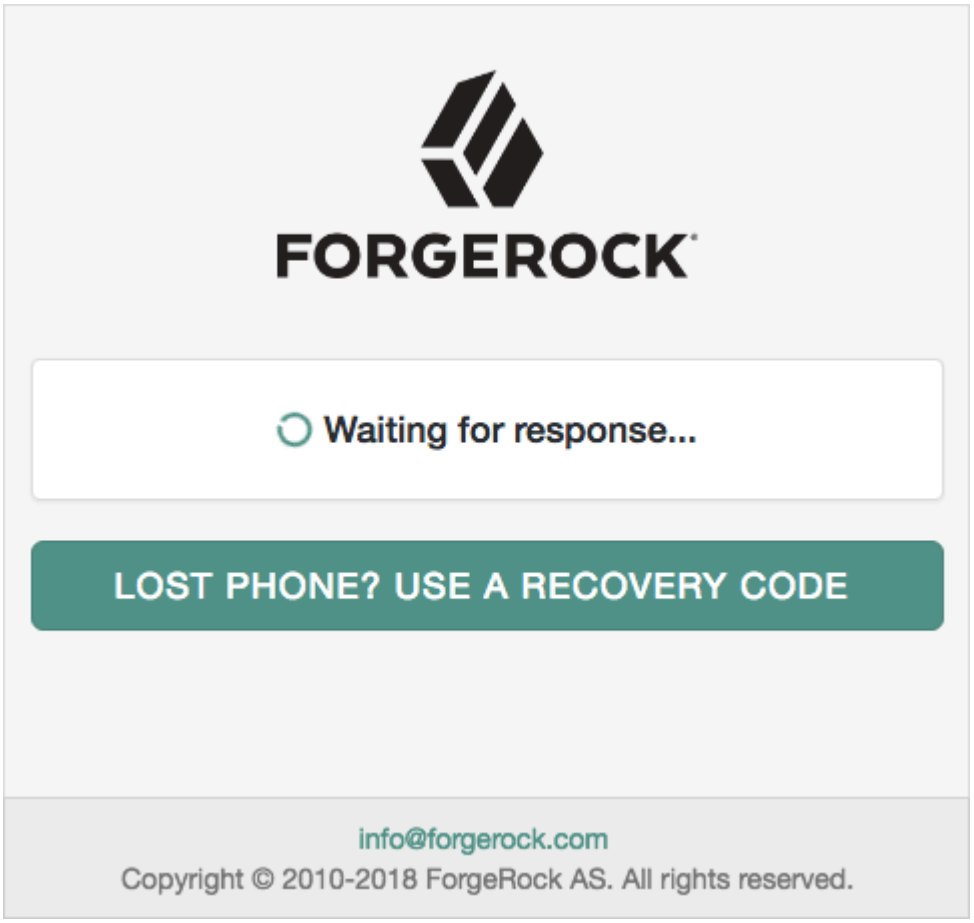
      A Retry Limit Decision node could be used here to constrain the number of times a new code is sent.

    - 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:



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. Present the user with information about registering their device.

   You can use the MFA Registration Options node, which has several built-in options, or a Page node with, for example, a Choice Collector node.

2. The user registers the device with the Push Registration node. After registration, the tree displays the recovery codes to the user for safekeeping.

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

- An Inner Tree Evaluator node may provide an alternative method of authentication. Otherwise, you may decide to allow the user to log in, as shown in the example.

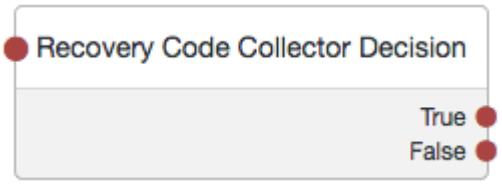
Recovery Code Collector Decision node

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

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 or Push 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 MFA nodes, when configured to generate recovery codes. 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

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:

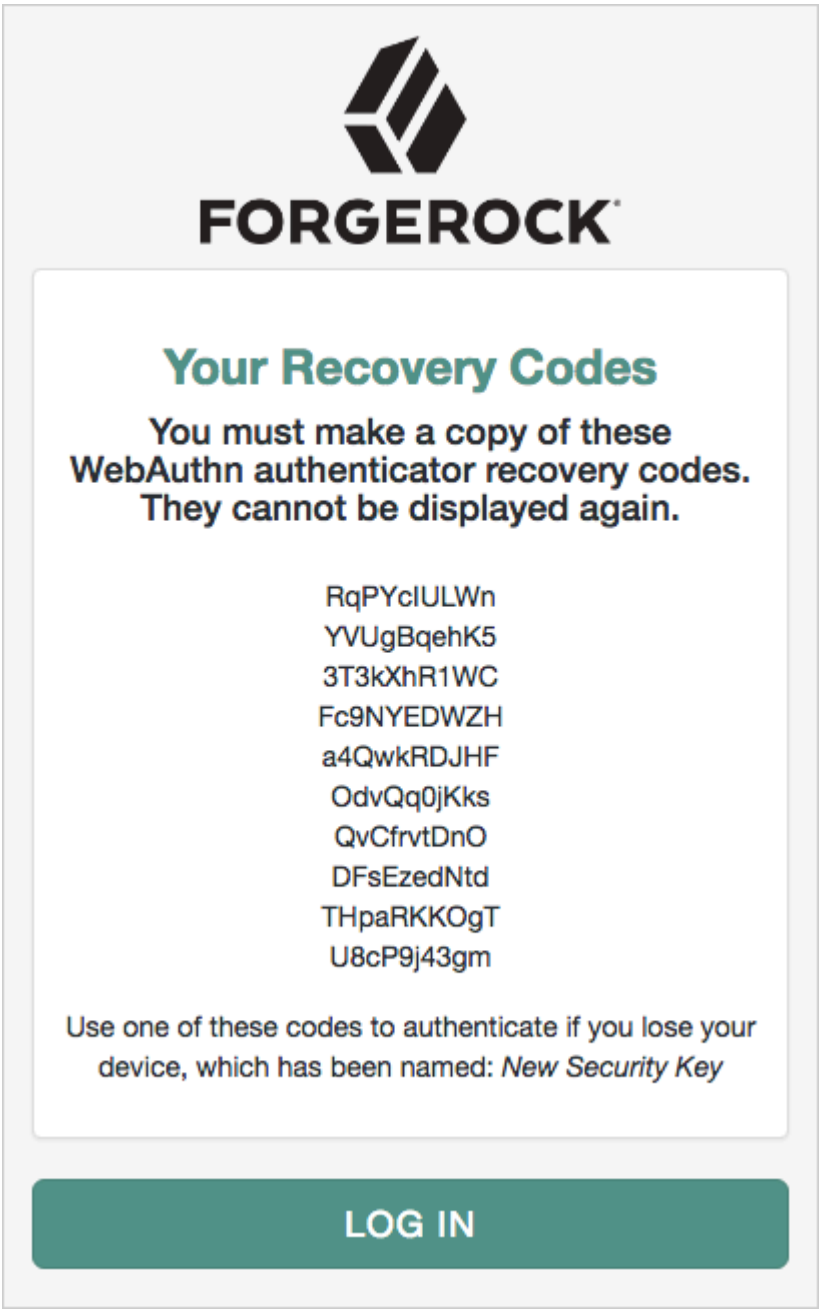


Figure 27. Example output of the Recovery Code Display node

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

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.

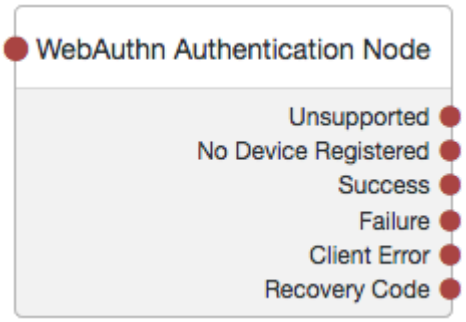
If the user's client encounters an issue when attempting to authenticate using the device, for example, if the timeout was reached, then tree evaluation continues along the `Client Error` outcome path. This outcome is used whenever the client throws a `DOMException`, as required by the Web Authentication: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



Properties

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.
Origin domains	Specifies a list of fully qualified URLs to accept as the origin of incoming requests. If left empty, AM accepts any incoming domain.

Property	Usage
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.
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.

Property	Usage
Username from device	Specifies whether AM requests that the device provides the username. When enabled, if the device is unable to store or provide usernames, the node will fail and results in the *Failure* outcome. For information on using this property for usernameless authentication with ForgeRock Go, see Configuring Usernameless Authentication with ForgeRock Go.
Return challenge as JavaScript	Specifies that the node returns its challenge as a fully encapsulated client-side JavaScript that interacts directly with the WebAuthn API, and auto-submits the response back. If disabled, the node returns the challenge and associated data in a metadata callback. A custom UI, for example an application using the ForgeRock SDKs, uses the information from the callback to interact with the WebAuthn API on AM's behalf.

Example

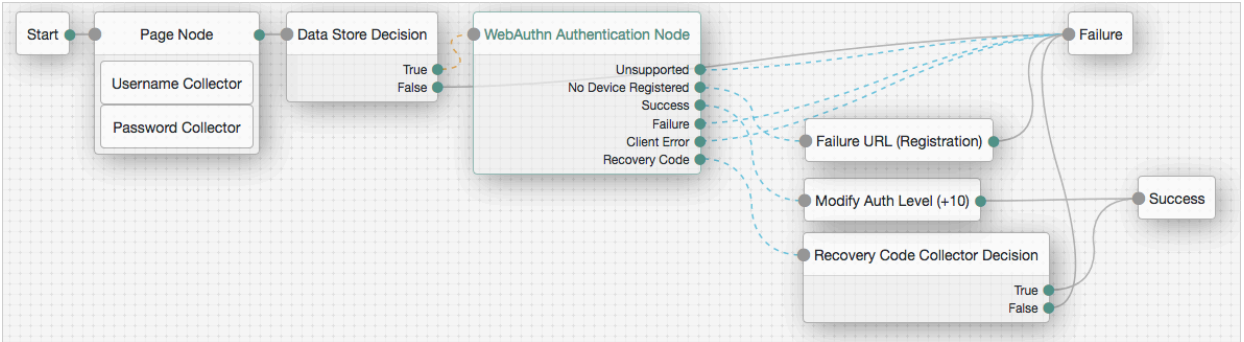


Figure 28. Example WebAuthn Authentication Tree (Standalone AM)

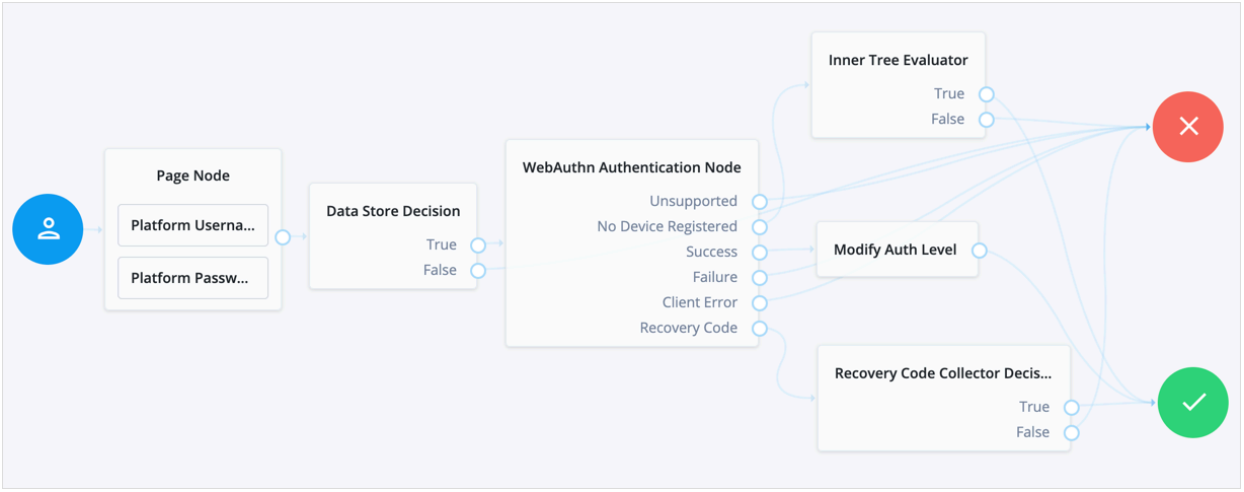


Figure 29. Example WebAuthn Authentication Tree (ForgeRock Identity Platform)

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 WebAuthn, 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 WebAuthn, 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:

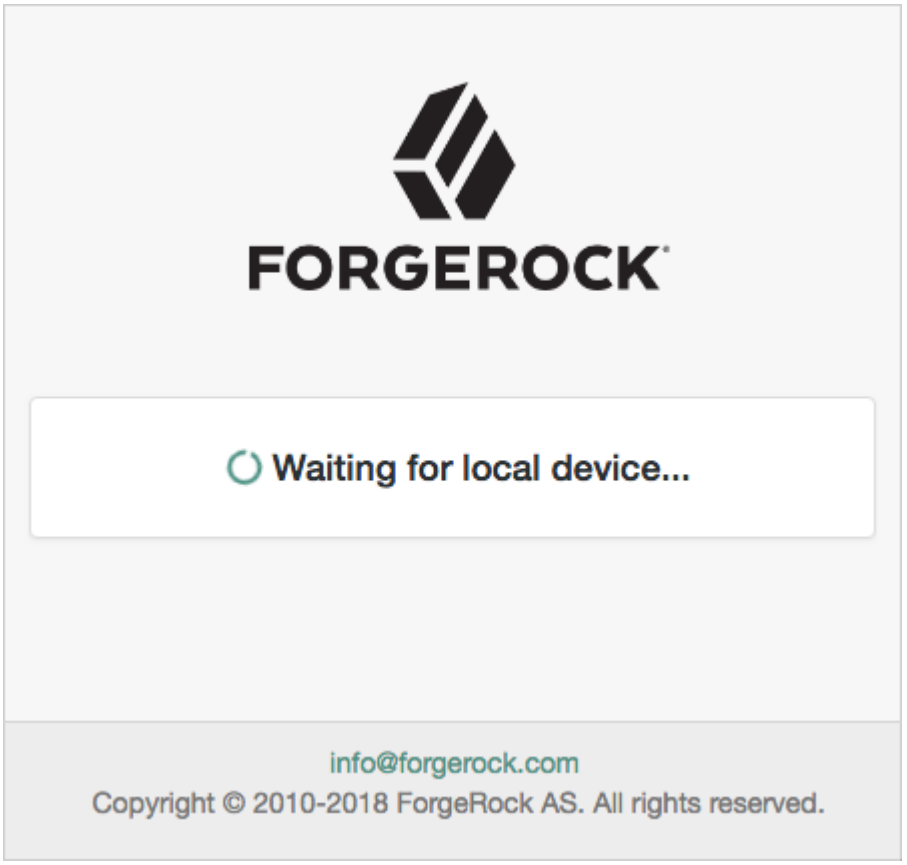


Figure 30. The WebAuthn Authentication node waiting for an authenticator (Standalone AM)

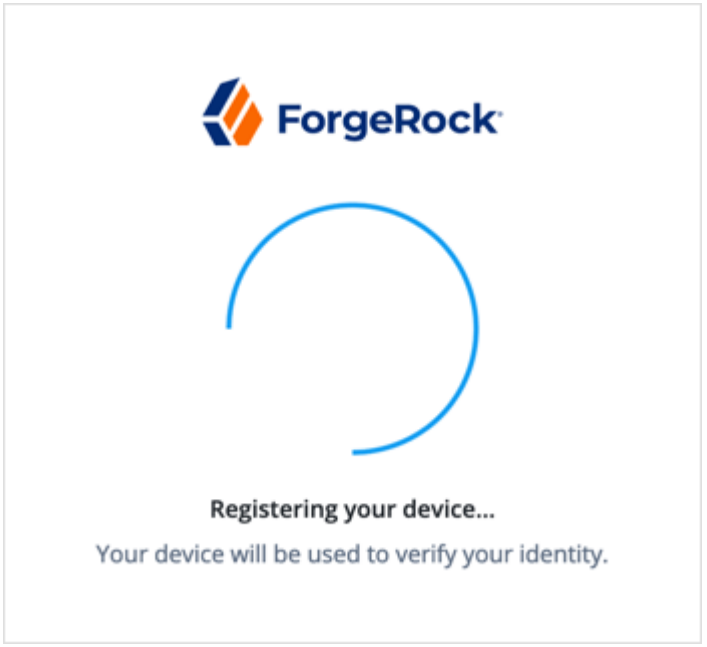


Figure 31. The WebAuthn Authentication node waiting for an authenticator (ForgeRock Identity Platform)

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 Device Storage node

The WebAuthn Device Storage node writes information about FIDO2 devices to a user's profile, so that they can subsequently authenticate using the device.

Use this node to store the device data that the WebAuthn Registration node places into the tree's transient state when its **Store device data in transient state** property is enabled.

If AM encounters an issue when attempting to save the device data to the user's profile; for example, the user has not been identified earlier in the tree, then tree evaluation continues along the `Failure` outcome path.

If the node successfully stores the device data to the user's profile, tree evaluation continues along the `Success` outcome path.

Properties

Property	Usage
Generate recovery codes	Specify whether WebAuthn device recovery codes should be generated. If enabled, recovery codes are generated and stored in the tree's transient state, and stored alongside the device profile. Use the Recovery Code Display node to display the codes to the user for safe keeping. IMPORTANT Generating recovery codes will overwrite all existing WebAuthn device recovery codes. Only the most recent set of recovery codes can be used for authentication if a device has been lost or stolen.

WebAuthn Registration node

The WebAuthn Registration authentication node allows users of supported clients to register FIDO2 devices for use during authentication.

AM interacts with FIDO2/WebAuthn capable browsers, for example `Chrome`, `Firefox` and `Microsoft Edge`. These browsers interact with CTAP2 authenticators, including U2F and FIDO2 Security Keys, and platforms such as `Windows Hello` or `MacOS TouchId`.

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

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.

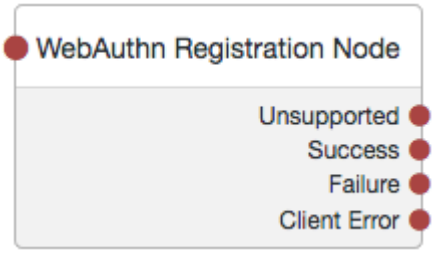
If the user's client encounters an issue when attempting to register using a device, for example, if the timeout was reached, then tree evaluation continues along the `Client`

`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 type as determined by the node's properties, tree evaluation continues along the `Success` outcome path.



Properties

Property	Usage
Relying party	Specify the name of the relying party entity that is registering and authenticating users by using WebAuthn. For example, `Example Inc.`.
Relying party identifier	Specifies the domain used as the relying party identifier during WebAuthn. 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.
Origin domains	Specifies a list of fully qualified URLs to accept as the origin of incoming requests. If left empty, AM accepts any incoming domain.

Property	Usage
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 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.

Property	Usage
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: • None • Android SafetyNet • Packed • FIDO U2F • TPM

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 the supported formats above. Specifically, AM *does not* currently support: • android-safetynet • android-key
Accepted signing algorithms	Specify the algorithms that authenticators can use to sign their assertions.

Property	Usage
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 labeled 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 labeled 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.

Property	Usage
Trust Store alias	Specifies the name of a secret store configured in the realm that contains CA-issued certificate chains, which can be used to verify attestation data provided by a device. The value is also appended to the string `am.authentication.nodes.webauthn.truststore.` to form the dynamic secret ID used to map the certificate chains. For more information, see Configuring WebAuthn Trust Anchors.
Enforce revocation check	Specifies whether to enforce certificate revocation checks. When enabled, then any attestation certificate's trust chain *MUST* have a CRL or OCSP entry that can be verified by AM during processing. When disabled, certificates are not checked for revocation. You must ensure expired or revoked certificates are manually removed.
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 WebAuthn-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. If you have enabled the **Store device data in transient state** and there are not saving the device data to the user's profile immediately, do not enable this property. Enable the **Generate recovery codes** property in the WebAuthn Device Storage node instead. IMPORTANT Generating recovery codes will overwrite all existing WebAuthn-specific recovery codes. Only the most recent set of recovery codes can be used for authentication if a device has been lost or stolen.

Property	Usage
Store data in transient state	Specify whether the information provided by the device to the node will be stored in the tree's transient state for later analysis by subsequent nodes, using the key `webauthnData`. In addition to the information provided by the device, the type of attestation achieved; for example, `BASIC`, `CA`, `SELF` and so on, will be stored in the tree's transient data, using the key `webauthnAttestationType`. WARNING The amount of data involved can be large. Only enable this option if you intend to analyze it.

Property	Usage
Store device data in transient state	Specify whether the information about the device required for WebAuthn is stored in the tree's transient state rather than saved immediately to the user's profile. Enable this option if you intend to make decisions in scripts, and have enabled the **Store data in transient state** property, and therefore do not want to register the device to the user until the outcome of the analysis is complete. IMPORTANT Do not alter the data whilst it is in the tree's transient state, nor when saved to a user's profile. Modifying the device data will likely cause the device to be unable to authenticate. Use the WebAuthn Device Storage node to write the device data to the user's profile when this option is enabled. When disabled, device data is written automatically to the user's profile when registration is successful.
Username to device	Specifies whether AM requests that the device stores the user's username. When enabled, if the device is unable to store or provide usernames, the node will fail and results in the *Failure* outcome. For information on using this property for usernameless authentication with ForgeRock Go, see Configuring Usernameless Authentication with ForgeRock Go.

Property	Usage
Shared state attribute for display name	Specifies a variable in tree's shared state that contains a display name for the user; for example, their full name, or email address. The value is written to devices alongside the username when the **Username to device** property is enabled, and helps the user select between the accounts they may have on their devices. If not specified, or the variable is not found in shared state, the user name is used. For information on using this property for usernameless authentication with ForgeRock Go, see Configuring Usernameless Authentication with ForgeRock Go.
Return challenge as JavaScript	Specifies that the node returns its challenge as a fully encapsulated client-side JavaScript that interacts directly with the WebAuthn API, and auto-submits the response back. If disabled, the node returns the challenge and associated data in a metadata callback. A custom UI; for example, an application using the ForgeRock SDKs, uses the information from the callback to interact with the WebAuthn API on AM's behalf.

Example

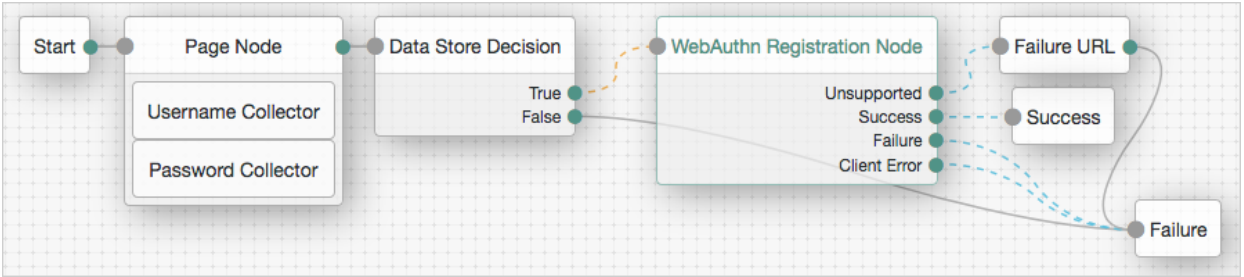


Figure 32. Example WebAuthn Registration Tree (Standalone AM)

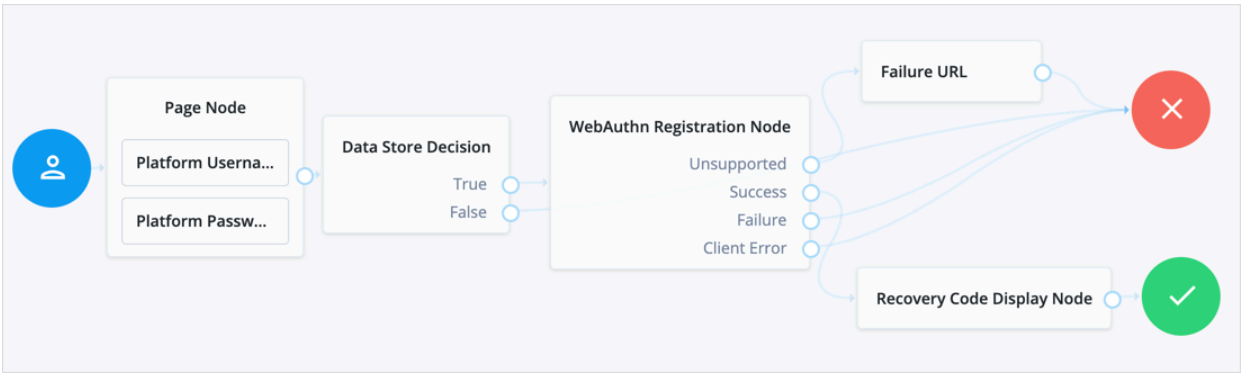


Figure 33. Example WebAuthn Registration Tree (ForgeRock Identity Platform)

The example tree above shows a possible implementation of a tree for registering WebAuthn 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 WebAuthn, the failure URL is altered, for example to redirect the user to a page explaining which clients and operating systems support WebAuthn.

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

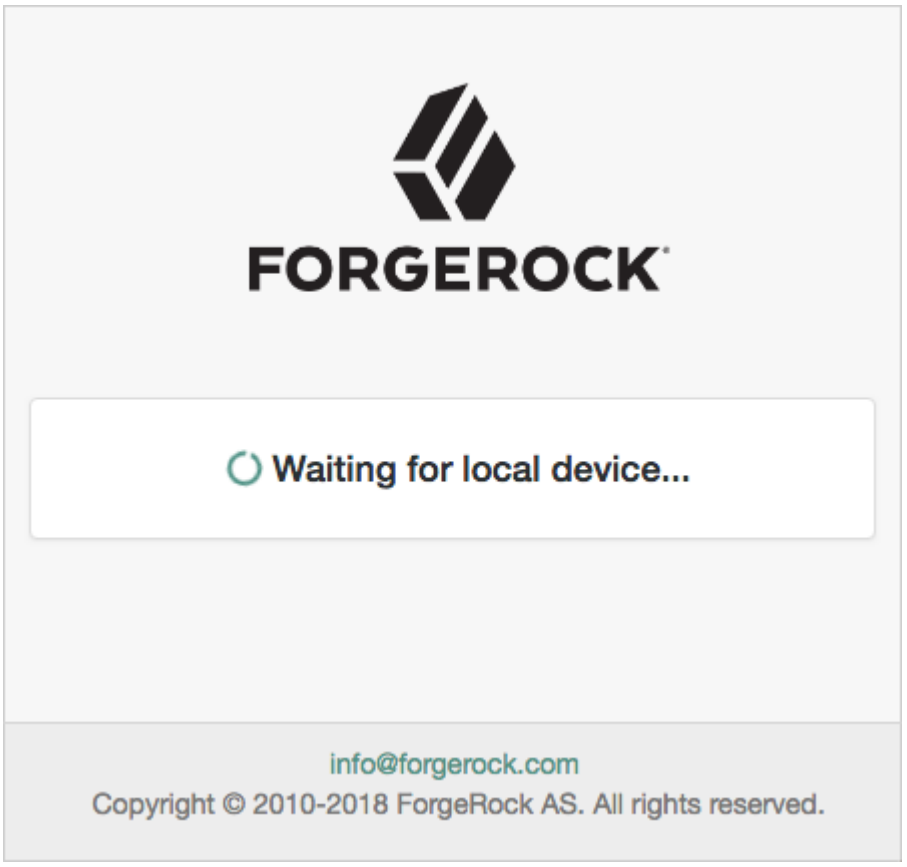


Figure 34. The WebAuthn Registration node waiting for an authenticator (Standalone AM)

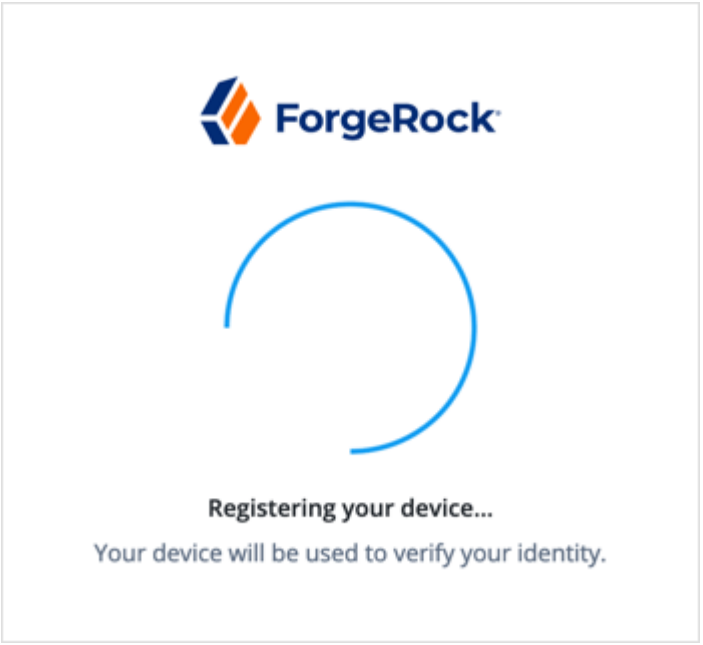


Figure 35. The WebAuthn Registration node waiting for an authenticator ForgeRock Identity Platform

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 click Rename.

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

Risk management authentication nodes

Use the following nodes to examine the perceived risk associated to the authentication and act on it:

Account Active Decision node

Checks if the account the user has entered is activated. This node relies on the tree's shared state to determine which account to check. Use this node to validate whether an account is currently activated, such as in login flows where an account may already be created, but not enabled until a later date.

For more information, see Account lockout for trees.

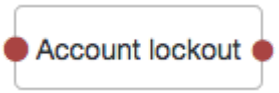
Properties

This node has no configurable properties.

Account Lockout node

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

For more information, see Account lockout for trees.



Properties

Property	Usage
Lock Action	Choose whether to `LOCK` or `UNLOCK` the authenticating user's account profile. 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 node with the Retry Limit Decision node to lock an account after a number of invalid attempts:

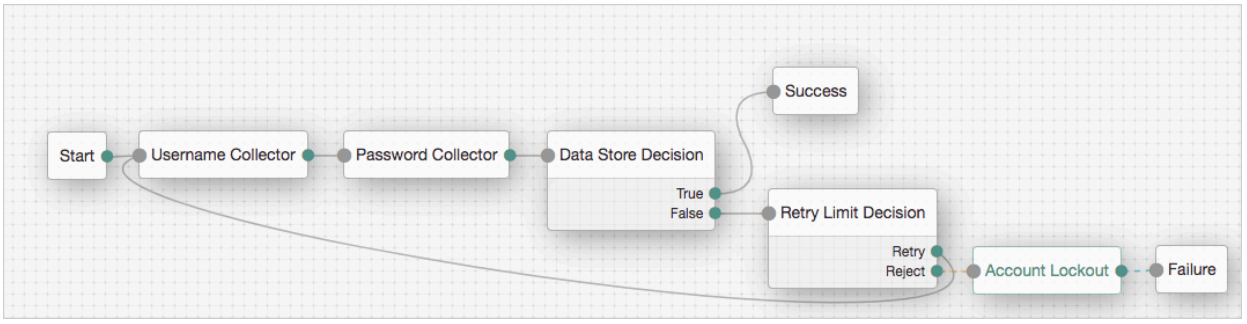


Figure 36. RetryLimit Tree With Account Lockout node (Standalone AM)

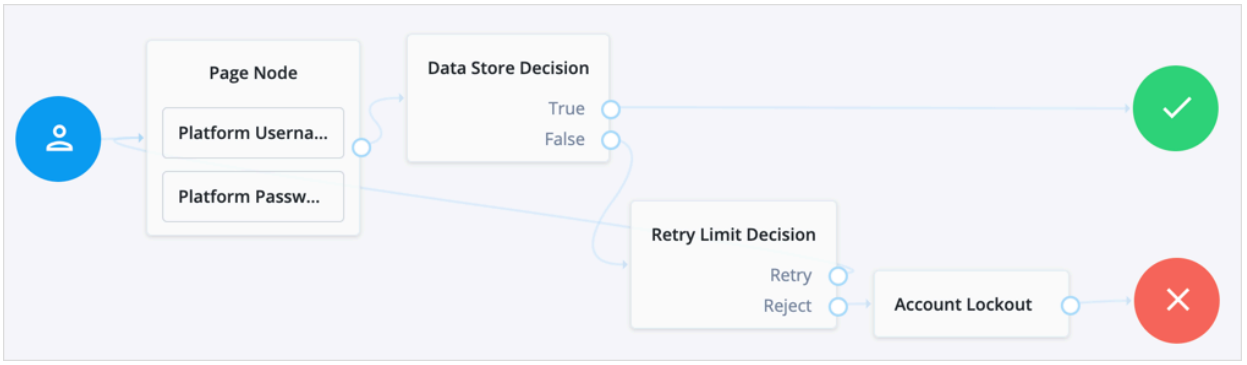


Figure 37. RetryLimit Tree With Account Lockout node (ForgeRock Identity Platform)

Auth Level Decision node

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

Properties

Property	Usage
Sufficient Authentication Level	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.

Modify Auth Level node

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

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

Properties

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.

CAPTCHA node

The CAPTCHA node implements Google's reCAPTCHA v2 and reCAPTCHA v3 widgets and hCaptcha's v1 widget, to add CAPTCHA support to authentication trees.

This node verifies the response token received from Google or hCaptcha and creates a CAPTCHA callback for the UI to interact with. The node has two outcomes—success and failure.

By default, the node is configured for Google's reCAPTCHA v2.

Properties

Property	Usage
CAPTCHA Site Key (required)	The CAPTCHA site key, provided by Google or hCaptcha when you sign up for access to the API.
CAPTCHA Secret Key (required)	The CAPTCHA secret key, provided by Google or hCaptcha when you sign up for access to the API.

Property	Usage
CAPTCHA Verification URL (required)	The URL used to verify the CAPTCHA submission. Possible values are: • Google: `https://www.google.com/recaptcha/api/siteverify` • hCaptcha: `https://hcaptcha.com/siteverify`
CAPTCHA API URL (required)	The URL of the JavaScript that loads the CAPTCHA widget. Possible values are: • Google: `https://www.google.com/recaptcha/api.js` • hCaptcha: `https://hcaptcha.com/1/api.js`
Class of CAPTCHA HTML Element	The class of the HTML element required by the CAPTCHA widget. Possible values are: • Google: `g-recaptcha` • hCaptcha: `h-captcha`
ReCaptcha V3 node	If you're using Google reCaptcha, specifies whether it's v2 or v3. Turn on for v3.
Score Threshold	If you're using Google reCAPTCHA v3, or hCaptcha, enter a Score Threshold. reCAPTCHA v3 and hCaptcha return a score for each user request, based on observed interaction with your site. Both facilities "learn" by observing real site traffic, so scores in a staging environment or in a production deployment that has just been implemented might not be very accurate. A score of 1.0 is likely a good user interaction, while 0.0 is likely to be a bot. The threshold you set here determines whether to allow or deny access, based on the score returned by Google or hCaptcha. You can generally start with a threshold of 0.5. For more information about score thresholds, see the Google documentation.

Example

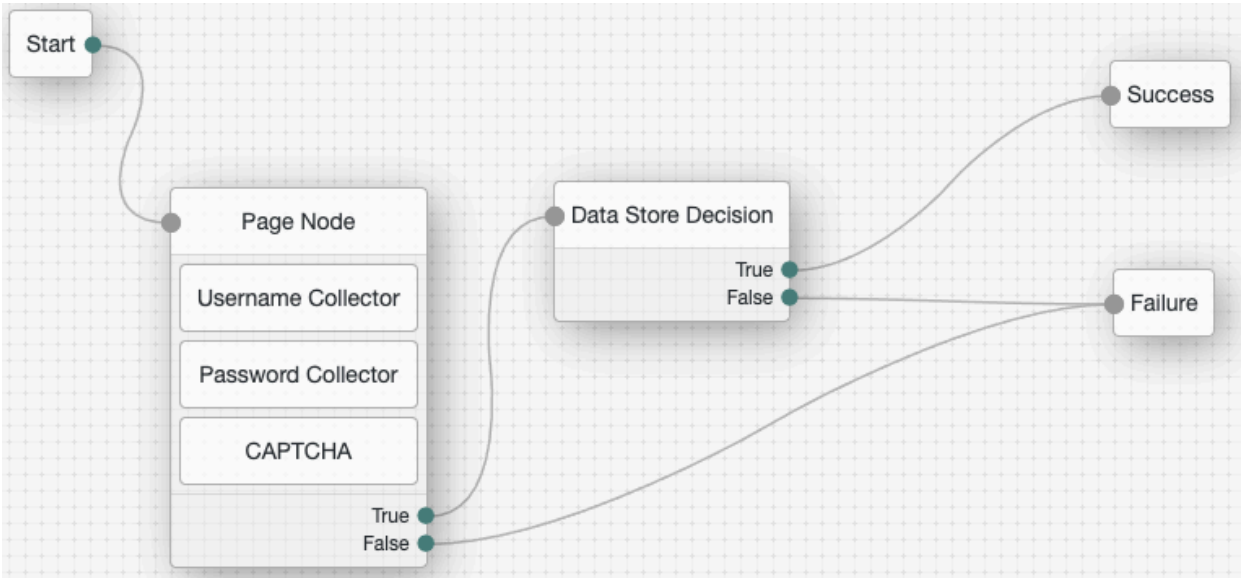


Figure 38. Example Tree With CAPTCHA node (Standalone AM)

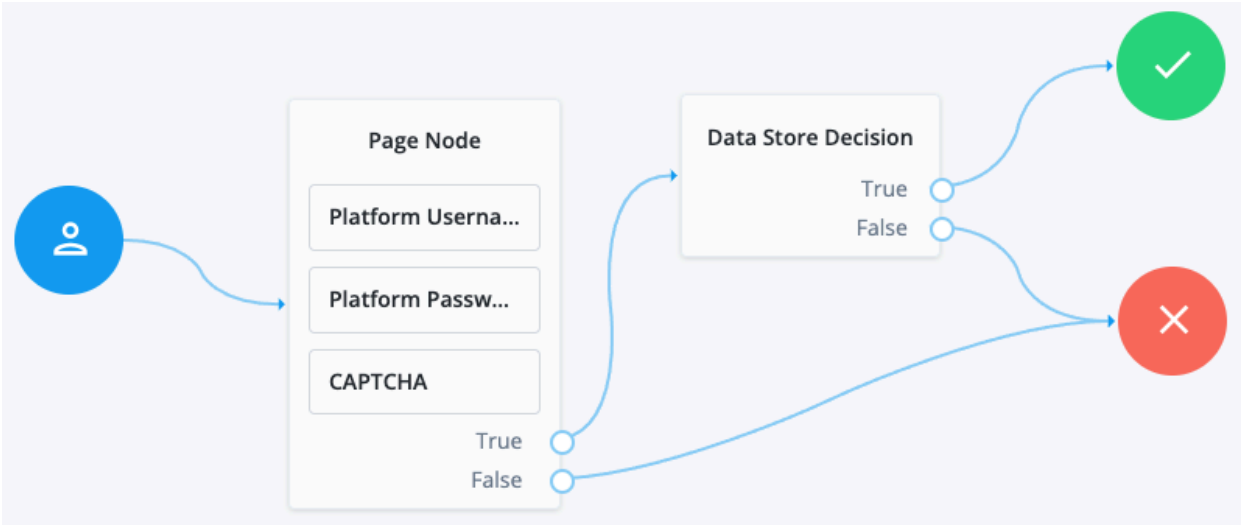


Figure 39. Example Tree With CAPTCHA node (ForgeRock Identity Platform)

Test the CAPTCHA node

ForgeRock provides a Postman collection to configure AM to test the CAPTCHA node. The Postman collection contains the queries to demonstrate the CAPTCHA node with reCAPTCHA V2, V3 and with hCaptcha. Before you start, set up a reCAPTCHA V2 and V3 site, and an hCaptcha site, and copy their site and secret keys.

1. Download and install Postman.
2. Download the ForgeRock CAPTCHA Collection.
3. Import the collection into Postman:
    - Select **File > Import ... > Upload Files**.
    - Select the CAPTCHA collection, and click **Open**, then click **Import**.
4. Change the collection variables to suit your environment:

- On the **Collections** tab, select the ForgeRock CAPTCHA Collection.
- Click on the **Variables** tab, and set the value of at least the following variables:
  - `URL_base`
  - `admin_password`
  - `demo_username`
  - `demo_password`
- Click **Update** to save your changes.

  You are ready to run the collection.

5. When the authentication trees have been created visit the following URLs in your browser to demonstrate the login flow for each CAPTCHA type:
   - URL_base/XUI/?realm=sub_realm&service=recaptchav3
   - URL_base/XUI/?realm=sub_realm&service=recaptchav2
   - URL_base/XUI/?realm=sub_realm&service=hcaptcha

Use the demo_username and demo_password to log in.

Behavioral authentication nodes

Use the following nodes to adjust the behavior of authentication trees:

Increment Login Count node

Increments the successful login count property of a managed object in IDM.

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

Use this node in conjunction with the Login Count Decision node. If you plan to track the number of logins, include this node in your login authentication flow, but you can safely omit it if you are not planning to use that functionality.

Properties

Property	Usage
Identity Attribute	The attribute used to identify the object in IDM.

Login Count Decision node

Triggers an action when a user's successful login count property reaches a specified number.

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

The action can either be triggered once, by setting the `interval` property to happen `AT` the set amount of successful login attempts; or set to occur `EVERY` time the specified number of additional successful login attempts occur.

Use this node in conjunction with the Increment Login Count node. The Increment Login Count node needs to be present in your login authentication flow for the Login Count Decision node to have the data necessary to trigger a decision.

Properties

Property	Usage
Interval	The type of interval the decision should trigger on. Valid types are `every` and `at`. `Every` refers to a recurring action that happens every specified number of successful logins, such as prompting a user to update their contact information every 30 days. `At` refers to an action that occurs once, after the specified number of successful logins. For example, prompting the user to set their communication preferences once they have logged in 10 times.
Amount	The amount (count) of logins the interval should trigger on.
Identity Attribute	The attribute used to identify the object in IDM.

Contextual authentication nodes

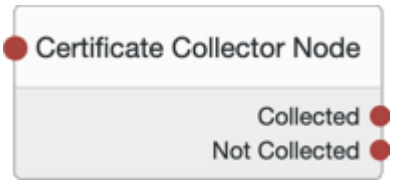
Use the following nodes to examine the authentication context and act on it:

Certificate Collector node

This node collects an X.509 digital certificate from the request coming from the authenticating user so that AM can use it as the user's credentials.

The tree continues through the `Collected` path if AM collects the digital certificate, and through the `Not Collected` path, otherwise.

To validate the certificate, add a Certificate Validation node to the tree.



Properties

Property	Usage
Certificate Collection Method	Specifies how AM should collect the certificate from the request. Possible values are: • `Request` . AM looks for the certificate in the request. Use this value if TLS termination happens at the container where AM runs. • `Header` . AM looks for the certificate in the HTTP header name specified in the **HTTP Header Name for the Client Certificate** property. Use this value if TLS termination happens in a proxy or load balancer placed in front of the container where AM runs. • `Either` . AM looks for the certificate in the request; if it cannot find it, AM looks for the certificate in the HTTP header specified in the **HTTP Header Name for the Client Certificate** property. Default: `Either`

Property	Usage
HTTP Header Name for the Client Certificate	Specifies the name of the HTTP header containing the certificate when the **Certificate Collection Method** property is configured to `Header` or `Either`. Default: No value specified.
Trusted Remote Hosts	Specifies a list of IP addresses trusted to supply certificates on behalf of the authenticating client, such as load balancers doing SSL termination. If no value is specified, AM will reject certificates supplied by remote hosts. If you specify the `any` value, AM will trust certificates on behalf of the authenticating client supplied by any remote host. Default: No value specified.

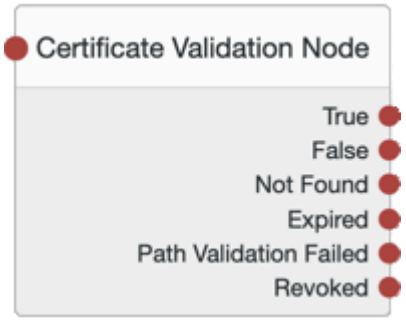
Certificate Validation node

This node validates a digital X.509 certificate collected by the Certificate Collector node.

The node has different outcomes, some of which are used depending on the configuration of the node:

- `True` : The node could validate the certificate.
- `False` : The node could not validate the certificate. The node will use this path when it cannot validate the certificate, but the cause is not managed by any of the other outcomes.
- `Not found` : The **Match Certificate in LDAP** property is enabled, but the certificate was not found in the LDAP store.
- `Expired` : The **Check Certificate Expiration** property is enabled, and the certificate has expired.
- `Path Validation Failed` : The **Match Certificate to CRL** property is enabled, and the certificate path is invalid.
- `Revoked` : The **OCSP Validation** property is enabled, and the certificate has been revoked.

When the outcome is `True` , append a Certificate User Extractor node to extract the values of the certificate and return them to AM.



Properties

Property	Usage
Match Certificate in LDAP	When enabled, AM matches the certificate collected with the one stored in an LDAP directory entry. This entry, and additional security-related properties, are defined later in the node. Default: Disabled
Check Certificate Expiration	When enabled, AM checks whether the certificate has expired. Default: Disabled
Subject DN Attribute Used to Search LDAP for Certificates	Specifies the attribute that AM will use to search the LDAP directory for the certificate. The search filter will also use the value of the Subject DN as it appears in the certificate. Default: `CN`
Match Certificate to CRL	When enabled, AM checks whether the certificate has been revoked according to a CRL in the LDAP directory. Related properties are defined later in the node. Default: Disabled.

Property	Usage
Issuer DN Attribute(s) Used to Search LDAP for CRLs	Specifies which attribute and value in the certificate Issuer DN AM will use to find the CRL in the LDAP directory. If only one attribute is specified, the LDAP search filter used 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)`. Specify several CLRs for the same CA issuer in a comma-separated list (`,`) where the names are in the same order as they occur in the subject DN. In this case, the LDAP search filter used is `(cn=attr1=attr1-value-in-subject-DN,attr2=attr2-value-in-subject-DN,…`, and so on. 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)`. Default: `CN`

Property	Usage
HTTP Parameters for CRL Update	Specifies parameters that AM will include in any HTTP CRL call to the CA that issued the certificate. If the client or CA contains the Issuing Distribution Point Extension, AM uses this information to retrieve the CRL from the distribution point. Add the parameters as key pairs of values in a comma-separated list (,). For example, `param1=value1,param2=value2` .
Cache CRLs in Memory	(LDAP distribution points only) When enabled, AM caches CRLs. Default: Enabled
Update CA CRLs from CRLDistributionPoint	When enabled, AM updates the CRLs stored in the LDAP directory store if the CA certificate includes either the `IssuingDistributionPoint` or the `CRLDistributionPoint` extensions. Default: Enabled
OCSP Validation	When enabled, AM checks the revocation status of certificates using the Online Certificate Status Protocol (OCSP). The AM instance must have internet access, and you must configure OSCP for AM by going to Configure > Server Defaults > Security > Online Certificate Status Protocol Check. Default: Disabled

Property	Usage
LDAP Server Where Certificates are Stored	Specifies the LDAP server that holds the certificates. Enter each server in the *`ldap-server:port`* format. 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`. To configure a secure connection, enable the Use SSL/TLS for LDAP Access property.
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 `am_server|base_dn`. For example, `openam.example.com|dc=example,dc=com` `openam1.test.com|dc=test,dc=com`.
LDAP Server Authentication User	Specifies the DN of the service account that AM will use to authenticate to the LDAP that holds the certificates. For example, `cn=LDAP User`. Default: `cn=Directory Manager`
LDAP Server Authentication Password	Specifies the password of the user configured in the **LDAP Server Authentication User** property.
Use SSL/TLS for LDAP Access	Specifies whether AM should use SSL/TLS to access the LDAP. When enabled, AM must be able to trust the LDAP server certificate. Default: Disabled

Example

The following is an example of how to use the certificate nodes. Note that all the failure outcomes of the Certificate Validation node are linked so that the user provides a username and password, but you could choose different authentication methods for each outcome.

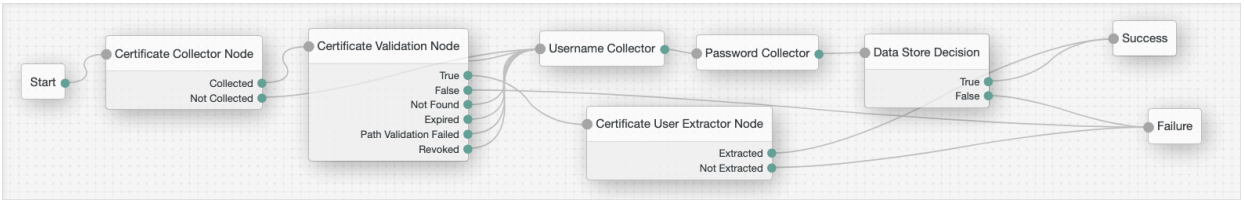


Figure 40. Certificate Validation Example (Standalone AM)

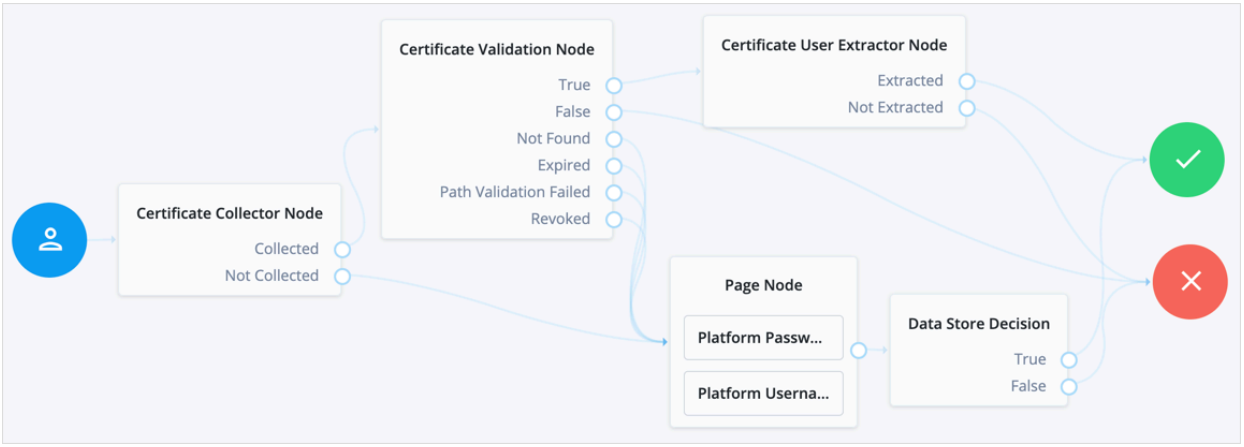


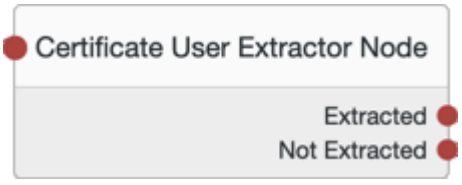
Figure 41. Certificate Validation Example (ForgeRock Identity Platform)

Certificate User Extractor node

This node extracts a value from the certificate collected by the Certificate Collector node, and searches for it in the identity store. The goal is to match the certificate with a user in the identity store.

The tree continues through the `Extracted` path if AM was able to match the certificate to a user in the identity store, and through the `Not Extracted` path otherwise.

The extracted value is stored in the `username` key in the shared state of the authentication tree.



Properties

Property	Usage
Certificate Field Used to Access User Profile	Specifies the field in the certificate that AM will use to search for the user in the identity store. Possible values are: • `Subject DN` • `Subject CN` • `Subject UID` • `Email Address` • `Other` • `None` If you select `Other`, provide an attribute name in the **Other Certificate Field Used to Access User Profile** property. Select `None` if you want to specify an alternate way of looking up the user profile in the **SubjectAltNameExt Value Type to Access User Profile** property. Default: `Subject CN`
Other Certificate Field Used to Access User Profile	Specifies a custom certificate field to use as the base of the user search.
SubjectAltNameExt Value Type to Access User Profile	Specifies how to look up the user profile. Possible values are: • `None`. AM uses the value specified in the **Certificate Field Used to Access User Profile** or the **Other Certificate Field Used to Access User Profile** properties when looking up the user profile. • `RFC822Name`. AM looks up for the user profile using the value of the `RFC822Name` field. • `UPN`. AM looks up the user profile as the User Principal Name attribute used in Active Directory. Default: `None`

Configuration Provider node

The Configuration Provider node creates a dynamic node configuration, based on the current node state. The node references a script that lets you build up a configuration map object with custom values, and provides this configuration map to the specified node type.

The node has all the outcomes of the specified node and an additional outcome, `Configuration failure`. Use the `Configuration failure` outcome to specify what should happen in the tree if the node is unable to build the configuration map, the configuration map is missing required values, or the configuration map is invalid.

Before you use the Configuration Provider node, create a script, of type `Config Provider`, that provides the functionality for the node. The map that you build in the script must be named `config`, and must be populated with the required configuration.

TIP

To obtain the list of required attributes that your script *must* set for a specific node type, see the API Explorer, and the endpoint `/realm-config/authentication/authenticationtrees/nodes/`*NodeType*`#_action_template`.

The following request returns the required attributes for the Message node:

For example:

```
$ curl \
--request POST \
--header "iPlanetDirectoryPro: <token>" \
"https://openam.example.com:8443/openam/json/realm-
config/authentication/authenticationtrees/nodes/MessageNode?
_action=template"
{
  "messageYes": {},
  "message": {},
  "messageNo": {}
}
```

This example creates a configuration map that pulls in the user's `username` from the node state to display a custom message, requesting the user to confirm that they are over 18:

```
config = {
    "message": {"en-GB": "Hi " + nodeState.get("username") + ".
Please confirm that you are over 18."},
    "messageYes": {"en-GB": "Confirm"},
```

```
    "messageNo": {"en-GB": "Deny"},
};
```

See the sample script for a reference to the bindings available to this script.

Example

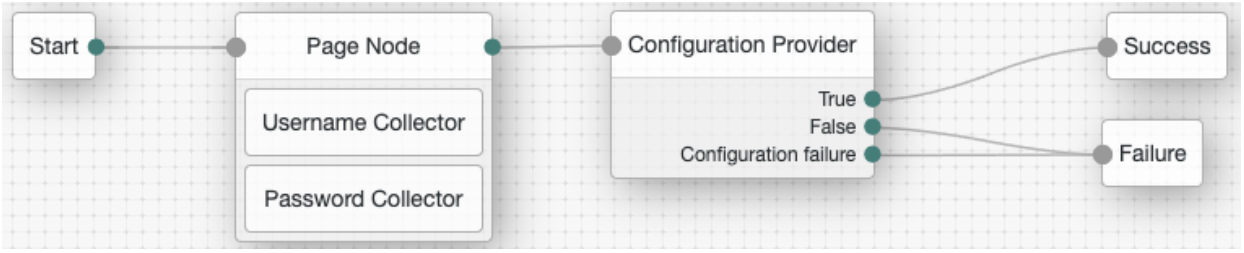


Figure 42. Example Tree With Configuration Provider Node (Standalone AM)

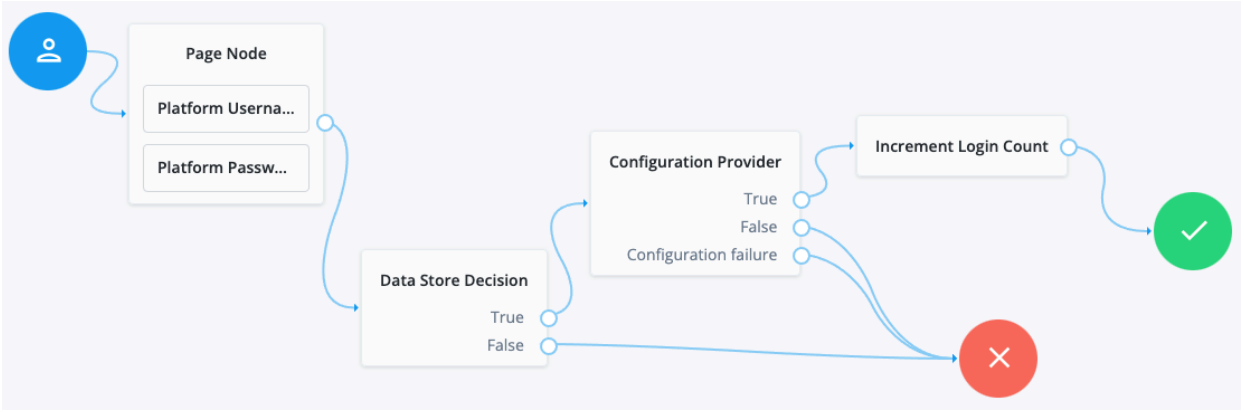
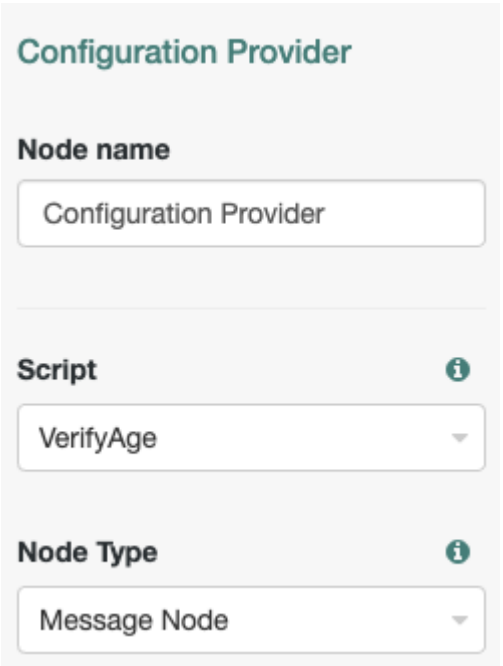


Figure 43. Example Tree With Configuration Provider Node (ForgeRock Identity Platform)

Referencing the script shown in the previous example, the configuration of this node would look something like this:



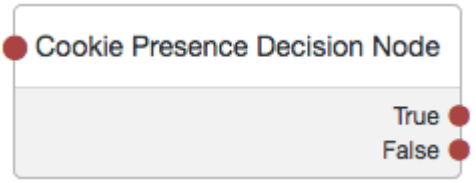
Properties

Property	Usage
Script	Select the Config Provider type script that you created for this node.
node Type	Select the type of node for which you are generating a configuration. The Configuration Provider node cannot provide configuration for all node types. The node type must have a fixed set of outcomes that cannot be changed by the node's configuration. (If the node's `OutcomeProvider` implements `StaticOutcomeProvider`, it will be picked up by the Configuration Provider node and will have an entry in the **node Type** dropdown list.)

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

Property	Usage
Name of Cookie	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.

Device Profile Collector

The Device Profile Collector authentication node gathers metadata about the device the user is authenticating with.

The node sends a `DeviceProfileCallback` callback. For more information, see Interactive Callbacks.

When used with the ForgeRock SDKs, the node can collect the following:

Device Metadata
: Information such as the platform, versions, device name, hardware information, and the brand of the device being used.

  The captured data is in JSON format, and stored in the authentication tree's shared state, in a variable named `forgeRock.device.profile`.

Device Location
: Provides the last known latitude and longitude of the device's location.

  The captured data is in JSON format, and stored in the authentication tree's shared state, in a variable named `forgeRock.device.location`.

  The collection of geographical information requires end user approval. This process is driven by a browser function. End users receive a message asking if sharing geographical location can be permitted. This function is not honored by the browser if the connection is not secure.

IMPORTANT

It is up to you what information you collect from users and devices.

You should always use data responsibly and provide your users appropriate control over data they share with you.

You are responsible for complying with any regulations or data protection laws.

In addition to the collected metadata, an `identifier` string in the JSON uniquely identifies the device.

Use this node with the Device Profile Save node when you want to create a trusted profile from the collected data. The trusted device profile can be used in subsequent authentication attempts; for example, with the Device Match and Device Profile Location Match node.

Properties

Property	Usage
Maximum Profile Size (KB)	Specifies the maximum accepted size, in kilobytes, of a device profile. If the collected profile data exceeds this size, authentication fails.
Collect Device Metadata	Specifies whether device metadata is requested.
Collect Device Location	Specifies whether device location is requested.
Message	Specifies an optional message to display to the user while the node collects the requested data. 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. Messages provided in the node override the defaults provided by AM. For information about customizing and translating the default messages, see Internationalization.

Device Match

The Device Match authentication node compares any collected device metadata with that stored in the user's profile.

Use this node alongside the Device Profile Collector node to determine if the authenticating user is on a previously saved, trusted device.

You can choose between two methods of comparison:

1. Built-in Matching

   The node handles the comparison and matching, and you can configure the acceptable variance, and specify a time frame that profiles are considered current.

2. Custom Matching

   Create scripts to do the comparison of captured device data against trusted device profiles.

   AM includes a template script that you can customize to your requirements. In the AM admin UI, go to Realms > *Realm Name* > Scripts, and click **Device Match Template - Decision node Script**.

   ForgeRock also provides a more complete sample script, as well as instructions for its use, and a development toolkit. Find these resources on GitHub, at: https://github.com/ForgeRock/forgerock-device-match-script.

You must establish the identity of the user in the tree before attempting to match device profiles.

Tree evaluation continues along the `True` path if the collected device profile matches a saved profile, within the configured variance; otherwise, tree evaluation continues along the `False` path.

If the user has no trusted device profiles, or the identity of the user has not been established, tree evaluation continues along the `Unknown Device` path.

Properties

Property	Usage
Acceptable Variance	Specify the maximum amount of device attribute differences that is still acceptable for a match.
Expiration	Specify the maximum age, in the number of days since being saved, that existing profiles can be considered for comparison. Device profiles that were saved to the user's profile before this time will not be compared to the collected metadata.

Property	Usage
Use Custom Matching Script	Specifies whether to use a custom script to compare the collected metadata with saved device profiles. The script type has to be *Decision node script for authentication trees*. NOTE When a custom matching script is used, the Acceptable Variance and Expiration properties are ignored.
Custom Matching Script	Specifies the custom script to use if the **Use Custom Matching Script** property is enabled. Only scripts of type *Decision node script for authentication trees* appear in the list.

Device Profile Save

The Device Profile Save authentication node persists collected device data to a user's profile in the identity store.

Use this node alongside the Device Profile Collector node when you want to reuse the collected data in future authentications; for example, with the Device Match and Device Profile Location Match node.

You must establish the identity of the user in the tree before attempting to save to their profile.

A user profile can contain multiple device profiles. Use the **Maximum Saved Profiles** property to configure the maximum number of device profiles to persist per user. Saving a device profile with the same identifier as an existing entry overwrites the original record, and does not increment the device profile count.

The *user UI* component of the platform UI displays saved device profiles to end users. Note that the Access Management UI **does not** display saved device profiles to end users.

You can manage device profiles with REST, by using the `/json/users/user/devices/profile` endpoint.

Use the AM API Explorer for detailed information about the parameters supported by the `/devices/profile` endpoint, and to test it against your deployed AM instance.

In the AM admin UI, select the Help icon, and then go to API Explorer > /users > /{user} > /devices > /profile.

Properties

Property	Usage
Device Name Variable	Specifies the name of a variable in the authentication tree's shared state that contains an alias label for the device profile.
Maximum Saved Profiles	Specify the maximum number of device profiles to save in a user's profile. When the maximum is reached, saving a new device profile replaces the oldest record.
Save Device Metadata	Specifies whether device metadata is saved to the user's profile.
Save Device Location	Specifies whether device location metadata is saved to the user's profile.

Device Profile Location Match

The Device Profile Location Match authentication node compares any collected device location metadata with that stored in the user's profile.

Use this node alongside the Device Profile Collector node to determine if the authenticating user's device is located within range of somewhere they have authenticated from, and saved, previously.

You must establish the identity of the user in the tree before attempting to match locations.

Tree evaluation continues along the `True` path if the collected location is within the specified range of saved location data; otherwise, tree evaluation continues along the `False` path.

If the user has no saved device profiles, or the identity of the user has not been established, tree evaluation continues along the `Unknown Device` path.

Properties

Property	Usage
Maximum Radius (km)	Specifies the maximum distance, in kilometers, that a device can be from a previously saved location. The distance is calculated point-to-point.

Device Geofencing

The Device Geofencing authentication node compares any collected device location metadata with the trusted locations configured in the authentication node.

Use this node alongside the Device Profile Collector node to determine if the authenticating user's device is located within range of configured, trusted locations.

Tree evaluation continues along the `Inside` path if the collected location is within the specified range of a configured trusted location; otherwise, tree evaluation continues along the `Outside` path.

Properties

Property	Usage
Trusted Locations	Specify the latitude and longitude of at least one trusted location. Separate the values with a comma; for example, `37.7910855,-122.3951663`.
Geofence Radius (km)	Specifies the maximum distance, in kilometers, that a device can be from a configured trusted location. The distance is calculated point-to-point.

Device Tampering Verification

The Device Tampering Verification authentication node specifies a threshold for deciding if the device has been tampered with; for example, if it has been rooted or jailbroken.

A score between zero and one is returned by the device, based on the likelihood that is has been tampered with or may pose a security risk. For example, an emulator scores the maximum of `1`.

Use this node alongside the Device Profile Collector node to retrieve the tampering score from the device.

Tree evaluation continues along the `Not Tampered` path if the device scores less than or equal to the configured threshold, otherwise tree evaluation continues along the `Tampered` path.

Properties

Property	Usage
Score Threshold	Specifies the score threshold for determining if a device has been tampered with. Enter a decimal fraction, between `0` and `1`; for example, `0.75`. The higher the score returned from the device, the more likely the device is jailbroken, rooted, or is a potential security risk. NOTE Emulators score the maximum; `1`.

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.

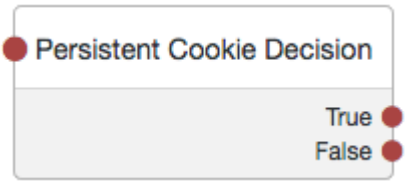
If the signature is valid, the node will decrypt the payload of the JWT by using the key pair specified in the Persistent Cookie Encryption Certificate Alias property. This property can be found at the global level by navigating to Configure > Authentication > Core Attributes > Security, or at the realm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

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

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, "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)."

Property	Usage

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

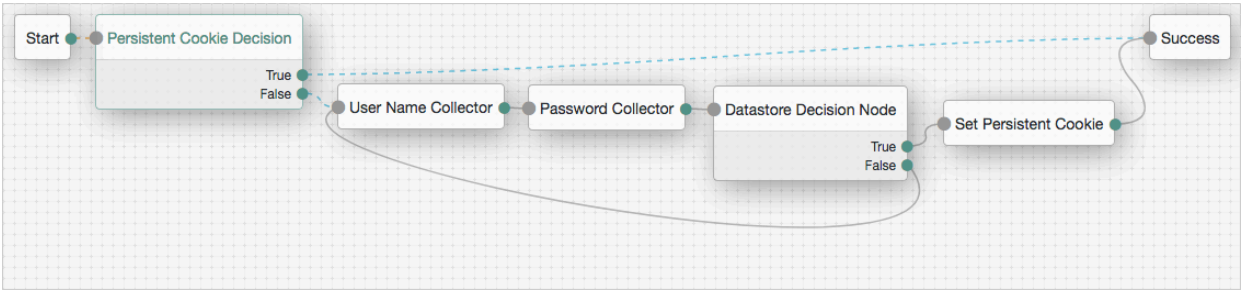


Figure 44. PersistentCookie Tree (Standalone AM)

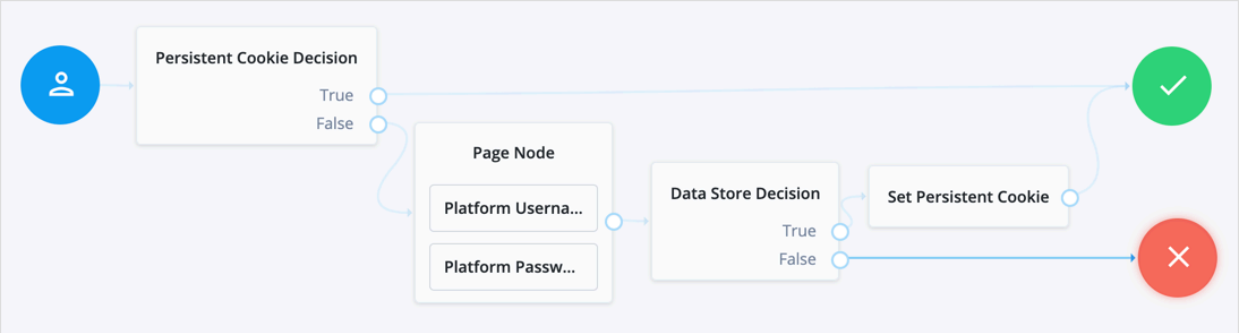


Figure 45. PersistentCookie Tree (ForgeRock Identity Platform)

Set Custom Cookie node

The Set Custom Cookie node lets you store an additional custom cookie in the client. The node uses the specified properties to create a cookie with a custom name and value, and optionally, sets attributes such as the cookie path, domain, expiry, and security flags.

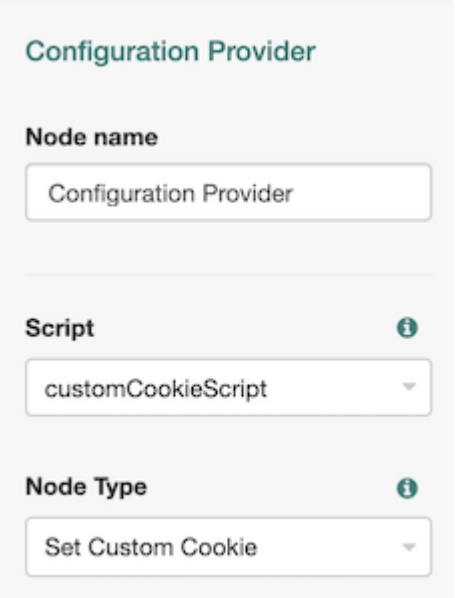
Tree evaluation continues along the single outcome path. The Set Custom Cookie is created when AM next returns to the client.

The Set Custom Cookie node can be used in conjunction with the Configuration Provider node to extend custom capabilities. For instance, create a `Config Provider` script to set custom static values or access values from the shared or transient state.

Include all the Set Custom Cookie node attributes in the configuration provider script's `config` map. This example shows how to set the attributes of the custom cookie to static values:

```
config = {
    "name": "testname",
    "value": "testvalue",
    "maxAge": "60",
    "domain": "am.exampledomain.com",
    "path": "/",
    "useSecureCookie": false,
    "useHttpOnlyCookie": false,
    "sameSite": "LAX"
};
```

Reference the script when you create a Configuration Provider node, and set the **node Type** to `Set Custom Cookie`. For example:



Example

This example uses a Set Custom Cookie node in a login tree.The custom cookie is set in the client browser after the user has successfully authenticated.

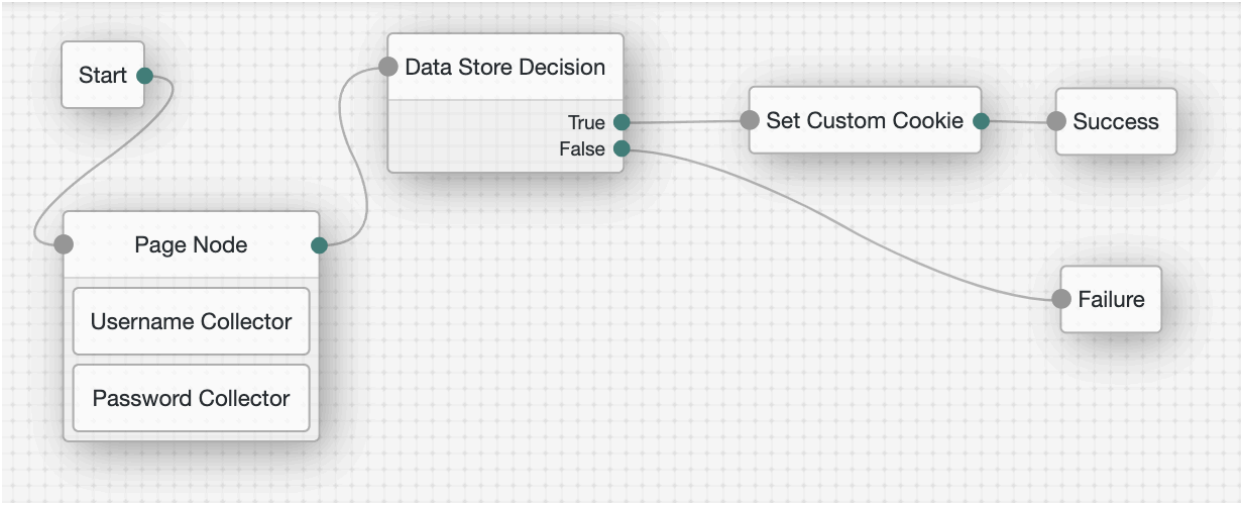


Figure 46. Login Tree With Set Custom Cookie node (Standalone AM)

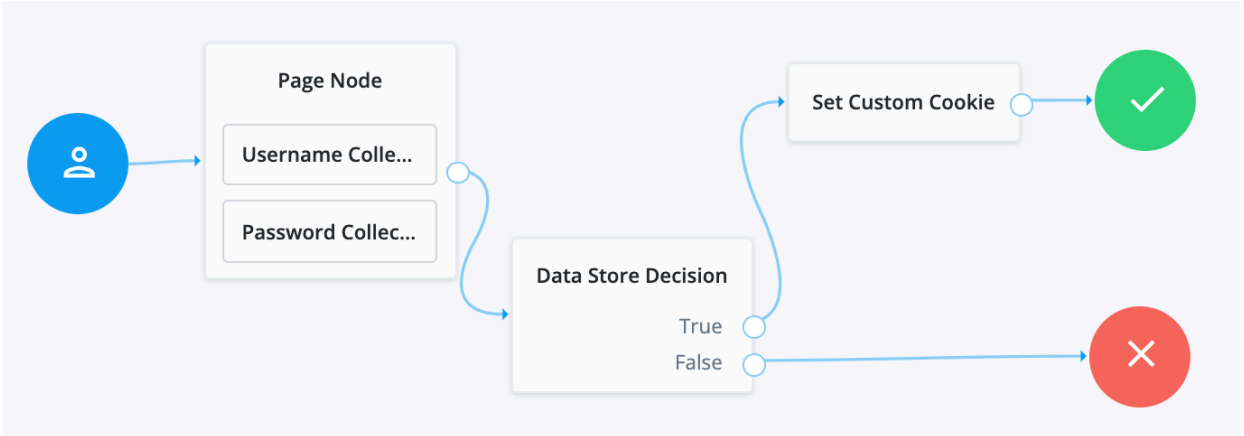


Figure 47. Login Tree With Set Custom Cookie node (ForgeRock Identity Platform)

Properties

Property	Usage
Custom Cookie Name	Sets the name of the custom cookie. The cookie name can contain any US-ASCII characters except for: space, tab, control, or a separator character (( ) < > @ , ; : " / [ ] ? = \ { }).
Custom Cookie Value	Sets the value of the custom cookie.
Max Age	(Optional). Specifies the length of time the custom cookie remains valid, in seconds. If that time is exceeded, the cookie is no longer valid. Both the `Max-Age` and `Expires` attributes are set in the cookie to increase compatibility with different browsers. If omitted, the cookie will expire at the end of the current session. The precise implementation of this is determined by the specific browser. See RFC 6265 for details.
Custom Cookie Domain	(Optional). Sets the domain that the custom cookie will be sent to.
Custom Cookie Path	(Optional). Sets the path of the custom cookie.
Use Secure Cookie	When enabled, adds the `Secure` flag to the custom 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 custom cookie. When the `HttpOnly` flag is included, the cookie is not accessible to scripts.

Property	Usage
Custom Cookie SameSite attribute	Sets the `SameSite` attribute of the custom cookie. The default value is `LAX`, to align with most modern browsers. For more information about cookies in AM, see SameSite cookie support in AM and IG.

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

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.

Property	Usage
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, "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)."

Property	Usage

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.

Federation authentication nodes

Use the following nodes to configure trees with federation capabilities, such as OAuth 2.0, social authentication, and account provisioning:

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.

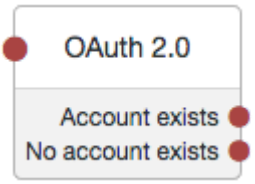
NOTE

NOTE

This node, and its related services, are deprecated. For an alternative, see Social authentication.

For information about the legacy/deprecated social authentication node and module implementations, see Social authentication in the *ForgeRock Access Management 7 Authentication and Single Sign-On Guide*.

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

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`

Property	Usage
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. Example: `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 UI. For example, `https://openam.example.com:8443/openam/XUI/`.

Property	Usage
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-`

Property	Usage

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.

Property	Usage
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" } }` You can create a mapper, such as `name.first_name=cn`.

Property	Usage
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.

Property	Usage
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. 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.

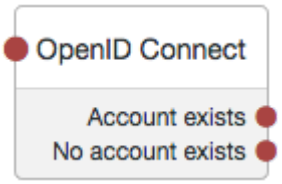
NOTE

This node, and its related services, are deprecated. For an alternative, see Social authentication.

For information about the legacy/deprecated social authentication node and module implementations, see Social authentication in the *ForgeRock Access Management 7 Authentication and Single Sign-On Guide*.

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.



Properties

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`

Property	Usage
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. If not specified, attributes are mapped from the claims returned by the `id_token` , and no call to a user profile endpoint is made. Example: `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 UI. For example `https://openam.example.com:8443/openam/XUI/` .

Property	Usage
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`

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. 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 click the Plus button (**+**).

Property	Usage

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 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.

Property	Usage
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`

Property	Usage
OpenID Connect Validation Type	Specifies how to validate the ID token received from the OpenID Connect provider. AM ignores keys specified in JWT headers, such as `jku` and `jwe`; 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 necessary 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`.
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.

Provision Dynamic Account node

The Provision Dynamic Account node provisions an account following successful authentication by a SAML2 authentication node or the Social Provider Handler node.

Accounts are provisioned using properties defined in the attribute mapper configuration of a social authentication or SAML2 authentication node earlier in the tree evaluation.

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

In addition to retrieving the password from the node state, the Provision Dynamic Account node gets the `realm` value, and `attributes` and `userNames` from `userInfo` in the shared state. It sets the `username` attribute in the node's shared state.

Properties

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.

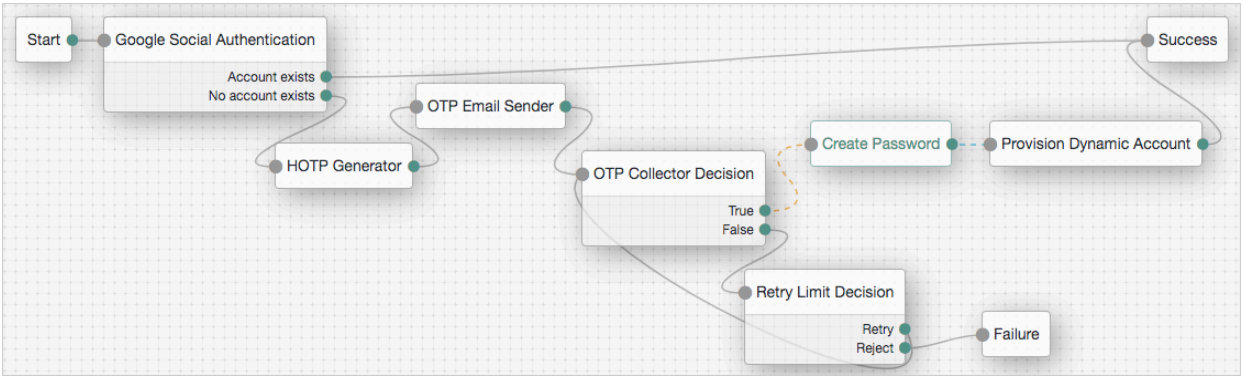


Figure 48. Google-DynamicAccountCreation Tree With Provision Dynamic Account node

Provision IDM Account node

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

NOTE

This node, and its related services, are deprecated. For an alternative, see Social authentication.

For information about the legacy/deprecated social authentication node and module implementations, see Social authentication in the *ForgeRock Access Management 7 Authentication and Single Sign-On Guide*.

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

Properties

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.

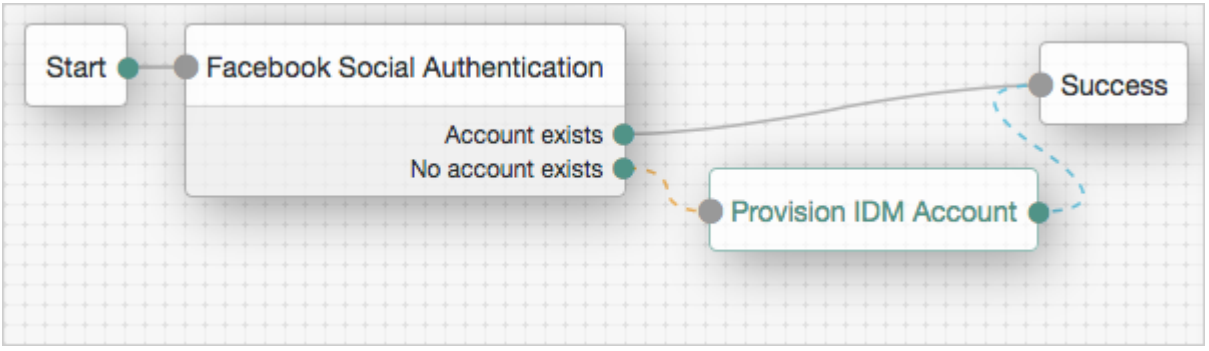


Figure 49. Facebook-ProvisionIDMAccount Tree With Provision IDM Account node

SAML2 Authentication node

This node lets you integrate SAML v2.0 SSO into an AM authentication tree. Use it when deploying SAML v2.0 single sign-on in integrated mode (SP-initiated SSO only).

If a user account is found that matches the federated account, tree evaluation continues along the "Account Exists" outcome. Otherwise, a matching account could not be found, and tree evaluation continues along the "No Account Exists" outcome.

If the node continues along the "Account Exists" or along the "No Account Exists" outcomes (in other words, if the node reaches the end of its processing without a failure), it sets the `successURL` parameter in the tree's shared state to the value of the `RelayState` parameter in the request, if any.

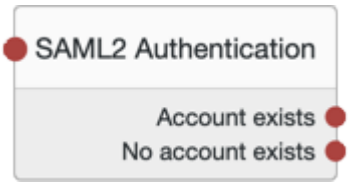
If the request does not provide a value, the node uses the default `RelayState` value configured in the SP.

You can dynamically provision an account on the SP if it does not exist, or you can link the remote account to a local account using the Write Federation Information node.

Before attempting to configure a SAML2 authentication node, ensure that:

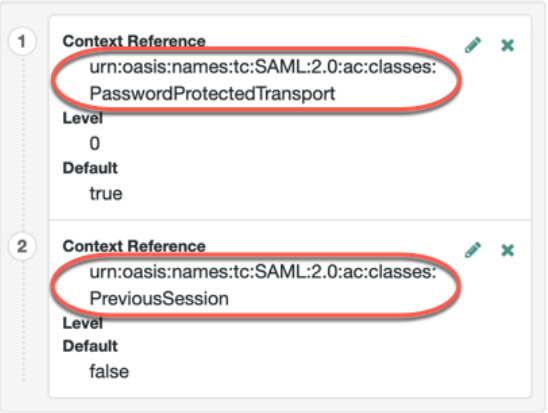
- You have configured a remote IdP and a hosted SP in a CoT in the same realm where the authentication node will be configured.
- The SP is configured for integrated mode. See SSO and SLO in Integrated Mode.

Properties



Property	Usage
IdP Entity ID	Specifies the name of the remote IdP.
SP MetaAlias	Specifies the local alias for the SP, in the format `/Realm Name/SP Name`.

Property	Usage
Allow IdP to Create NameID	Specifies whether the IdP should create a new identifier for the authenticating user if none exists. For detailed information, see the section on the `AllowCreate` property in SAML Version 2.0 Errata 05. Default: `Enabled`
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: `minimum`

Property	Usage
Authentication Context Class Reference	(Optional) 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. In the AM admin UI, go to Realms > *Realm Name* > Applications > Federation > Entity Providers > *Service Provider Name* > Assertion Content > Authentication Context.  When specifying multiple authentication context classes, use the `|` character to separate the classes. For example: `urn:oasis:names:tc:SAML:2.0:ac:classes:Password|urn:oasis:names:tc:SAML:2.0:ac:classes:TimesyncToken`

Property	Usage
Authentication Context Declaration Reference	(Optional) Specifies one or more URIs that identify authentication context declarations. 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.
Request Binding	Specifies the format the SP will use to send the authentication request to the IdP. Valid values are `HTTP-Redirect` and `HTTP-POST`. Default: `HTTP-Redirect`
Response Binding	Specifies the format the IdP will use to send the response to the SP. Valid values are `HTTP-POST` and `HTTP-Artifact`. Default: `HTTP-Artifact`
Force IdP Authentication	Specifies whether the IdP forces authentication or if it can reuse existing security contexts. Default: Disabled
Passive Authentication	Specifies whether the IdP uses 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. Default: Disabled

Property	Usage

NameID Format	Specifies the SAML name ID format that will be requested in the SAML authentication request. For example:
	`urn:oasis:names:tc:SAML:2.0:nameid-format:persistent`
	`urn:oasis:names:tc:SAML:2.0:nameid-format:transient`
	`urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified`
	Default:
	`urn:oasis:names:tc:SAML:2.0:nameid-format:persistent`

For examples, see SSO and SLO in integrated mode.

Social Facebook node

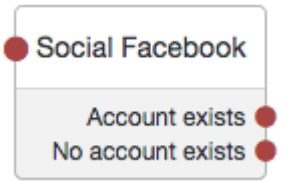
The Social Facebook authentication node is a duplicate of the OAuth 2.0 node, preconfigured to work with Facebook. Only the `Client ID` and `Client Secret` are required to be populated.

NOTE

This node, and its related services, are deprecated. For an alternative, see Social authentication.

For information about the legacy/deprecated social authentication node and module implementations, see Social authentication in the *ForgeRock Access Management 7 Authentication and Single Sign-On Guide*.

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). 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.

Property	Usage
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 UI. 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.
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`

Property	Usage
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: `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-`

Property	Usage
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`.

Property	Usage
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: `{ "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`.

Property	Usage
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.

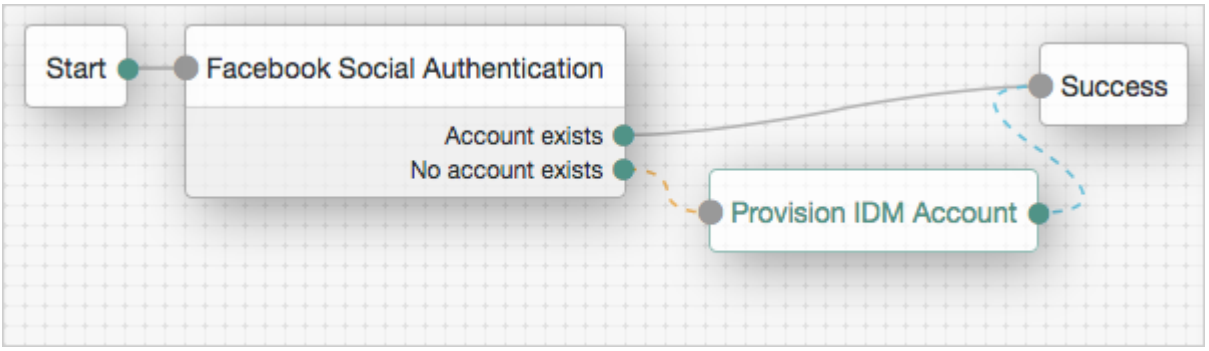


Figure 50. Facebook-ProvisionIDMAccount Tree With Provision IDM Account node

Social Google node

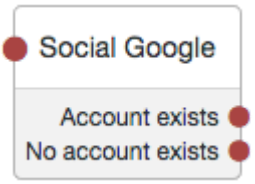
The Social Google authentication node is a duplicate of the OAuth 2.0 node, preconfigured to work with Google. Only the `Client ID` and `Client Secret` are required to be populated.

NOTE

This node, and its related services, are deprecated. For an alternative, see Social authentication.

For information about the legacy/deprecated social authentication node and module implementations, see Social authentication in the *ForgeRock Access Management 7 Authentication and Single Sign-On Guide*.

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.

Property	Usage
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`
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`.

Property	Usage
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 UI. 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.
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`

Property	Usage
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: `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper|iplanet-am-user-alias-list|google-`

Property	Usage
Account Mapper Configuration	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 *`provider-attr=local-attr`*. Default: `sub=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`.

Property	Usage
Attribute Mapper Configuration	Map of Google user account attributes to local user profile attributes, with values in the form *`provider-attr=local-attr`*. Default: `sub=uid`, `name=cn`, `given_name=givenName`, `family_name=sn`, `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: `{ "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`.

Property	Usage
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.

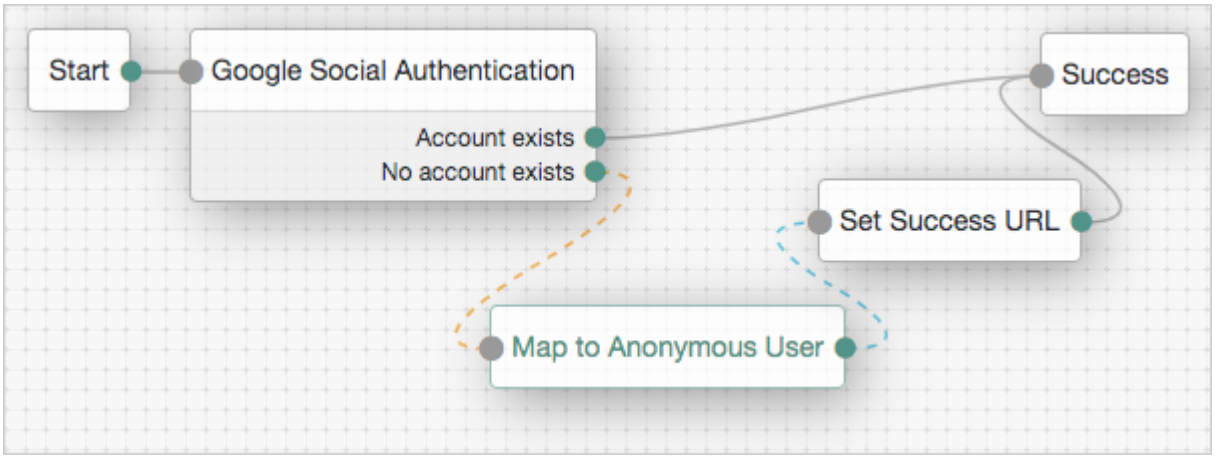


Figure 51. Google-AnonymousUser Tree With Anonymous User Mapping node

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.

NOTE

This node, and its related services, are deprecated. For an alternative, see Social authentication.

For information about the legacy/deprecated social authentication node and module implementations, see Social authentication in the *ForgeRock Access Management 7 Authentication and Single Sign-On Guide*.

Properties

This node has no configurable properties.

Social Provider Handler node

This node is used alongside the Select Identity Provider node to enable use of the Social Identity Provider Service.

It takes the provider selection from the Select Identity Provider node and attempts to authenticate the user with that provider. It then collects relevant profile information from the provider and returns the user to the flow, and transforms that profile information into attributes AM can use.

Properties

Property	Usage
Transformation Script	A script that transforms a normalized social profile to an identity or managed object. Select `Normalized Profile to Identity`, or any other script you have created for this purpose.
Username Attribute	ForgeRock Identity Platform deployments only.
Client Type	Specify the client type you are using to authenticate to the provider. Use the default, `BROWSER`, when making use of the ForgeRock-provided user interfaces, or the ForgeRock SDK for JavaScript. This causes the node to return the RedirectCallback. Select `NATIVE` if you are using the ForgeRock SDKs for Android or iOS. This causes the node to return the IdPCallback.

Write Federation Information node

This node creates a persistent link between a remote IdP account and a local account in the SP, if none exists yet. If a transient link exists, it is persisted. Existing account links with different IdPs are not lost.

Use this node alongside the SAML2 Authentication node, and ensure that the NameID Format is `persistent`.

Properties

This node has no configurable properties.

For examples, see SSO and SLO in Integrated Mode.

Identity management authentication nodes

Use the following nodes to perform identity management during an authentication tree flow, such as mapping anonymous users to a session.

Accept Terms and Conditions node

This node prompts the user to accept the currently active Terms and Conditions.

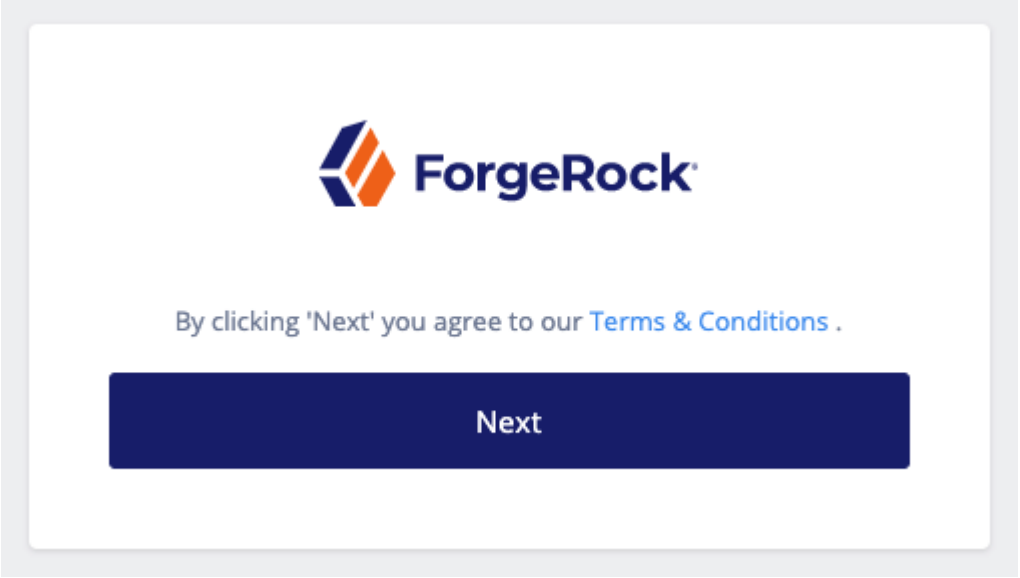
NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

You set up Terms and Conditions in the Platform UI. For more information, see Terms & Conditions in the platform documentation.

This node is used in a registration tree, or combined with the Terms and Conditions Decision node in a progressive profile or login tree.

Note that there is no failure path for this node. The user must accept the Terms and Conditions in order to proceed:



Properties

This node has no configurable properties.

Example

In a progressive profile tree, the Accept Terms and Conditions node is used after the Terms and Conditions Decision node. If the user has not accepted the latest version of

the Terms and Conditions, they are taken to a page notifying them that proceeding indicates accepting the current Terms and Conditions.

If the user clicks next, the acceptance response is stored in IDM.

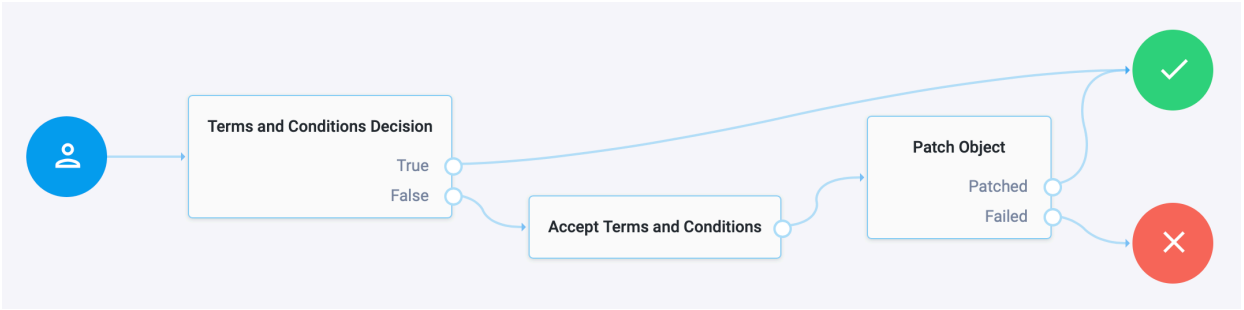


Figure 52. Example Tree With Accept Terms and Conditions node

Anonymous User Mapping node

The Anonymous User Mapping node lets users 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.

IMPORTANT

The anonymous user status is `Inactive` by default. If you use the default anonymous user in this node, you must change the user status to `Active`.

Properties

Property	Usage
Anonymous User Name	Specifies the username of an account that represents anonymous users. This user must already exist in the realm, and its user status must be `Active`.

Example

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

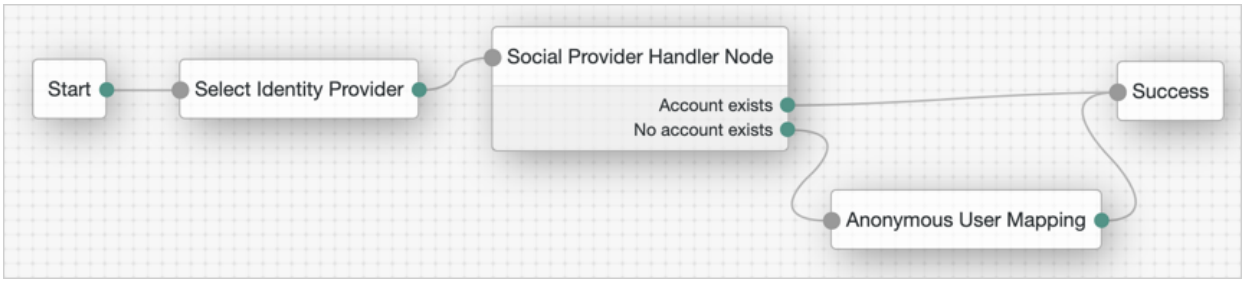


Figure 53. Social Identity Provider With Anonymous User Mapping node (Standalone AM)

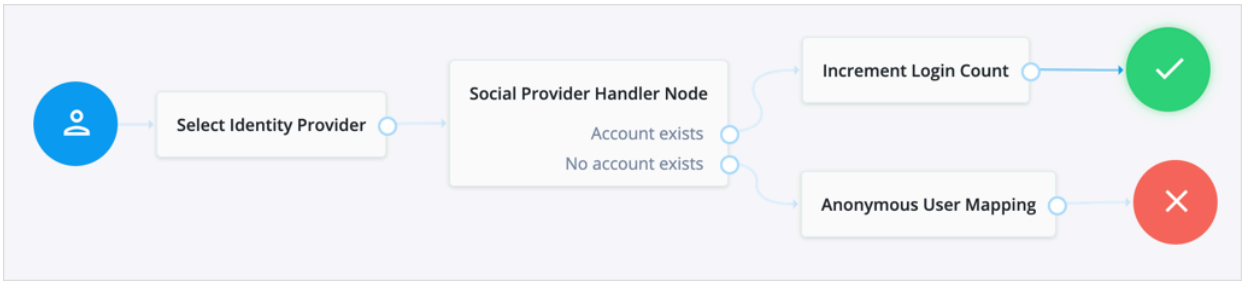
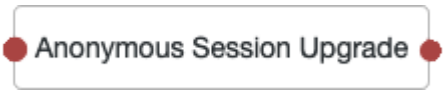


Figure 54. Social Identity Provider With Anonymous User Mapping node (ForgeRock Identity Platform)

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

This node has no configurable properties.

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.

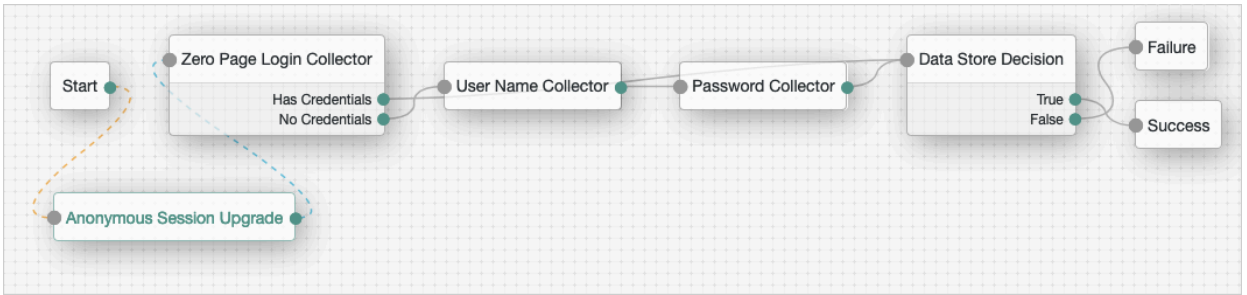


Figure 55. Example Tree With Anonymous Session Upgrade node (Standalone AM)

Figure 56. Example Tree With Anonymous Session Upgrade node (ForgeRock Identity Platform)

Attribute Collector node

The Attribute Collector node is used to collect the values of attributes for use elsewhere in a tree, such as collecting user information to populate a new account in a registration tree.

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

To request a value, the attribute must be present in the IDM schema of the **Identity Object** configured in the tree. This node supports three types of attributes: `string`, `boolean`, and `number`.

The node configuration allows the admin to specify if the attributes are required to continue, and if they should be subject to validation through IDM's policy filter.

You can place the node anywhere in your authentication tree, or within a page node.

Properties

Property	Usage
Attributes to Collect	A list of the attributes you wish to collect, based on the attributes found in the IDM schema for the identity object configured in the tree.
All Attributes Required	When enabled, all attributes collected in this node are required in order to continue.

Property	Usage
Validate Input	When enabled, the content input in this node should be validated against IDM policy settings specified in the IDM schema.
Identity Attribute	The attribute used to identify the object in IDM.

Attribute Present Decision node

Checks if an attribute is present on an object, regardless of whether the field is private. Use this to verify an attribute is present, without needing to know the value of the attribute itself.

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

A good use case is during an update password flow, where you want to check if the account has a password (rather than no password and logging in through a social identity) before continuing.

This node is similar to the Attribute Value Decision node when that node is set to use the `PRESENT` operator, except it cannot return the value of the attribute, but can work with private attributes.

Properties

Property	Usage
Present Attribute	The object attribute to verify is present in the IDM object. This can be an otherwise private attribute, such as `password`.
Identity Attribute	The attribute used to identify the object in IDM.

Attribute Value Decision node

Verifies that the user's specified attribute satisfies a specific condition.

NOTE

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

Use this node to check whether an attribute's expected value is equal to a collected attribute value, or to validate that a specified attribute has been collected (regardless of the value of that attribute).

For example, to validate that a user filled out the country attribute when registering, set the comparison operation to `PRESENT`, and the comparison attribute to `country`.

If you instead need to ensure the country attribute is set to the United States, set the comparison operation to `EQUALS`, the comparison attribute to `country`, and the comparison value to `United States`.

Use Attribute Present Decision node instead when you need to check for the presence of a private attribute (such as, `password`).

Properties

Property	Usage
Comparison Operation	The operation to perform on the object attribute; `PRESENT` checks for existence of an attribute, `EQUALS` checks if the object's attribute value equals the configured comparison value.
Comparison Attribute	The object attribute to compare.
Comparison Value	This property is only relevant when using the `EQUALS` comparison operation, and is the value to compare the object's attribute value to.
Identity Attribute	The attribute used to identify the object in IDM.

Create Object node

The Create Object node is used to create a new object in IDM based on information collected during an auth tree flow, such as user registration.

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

Any managed object attributes that are marked as required in IDM will need to be collected during the auth tree flow in order for the new object to be created.

Properties

Property	Usage
Identity Resource	The type of IDM managed identity resource object that this node will create. It must match the identity resource type for the current tree. TIP To check for the available managed identity resource types, go to the IDM admin UI, and open the **Manage** drop-down list, at the upper right corner of the screen. Identity managed object types are preceded by the icon.

Create Password node

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

NOTE

This node, and its related services, are deprecated. For an alternative, see Social authentication.

For information about the legacy/deprecated social authentication node and module implementations, see Social authentication in the *ForgeRock Access Management 7 Authentication and Single Sign-On Guide*.

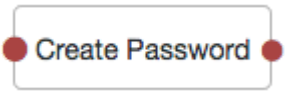
The social identity provider will not provide a user's password. Use this node to provide a password to complete the user's credentials before provisioning an account.

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

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

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.

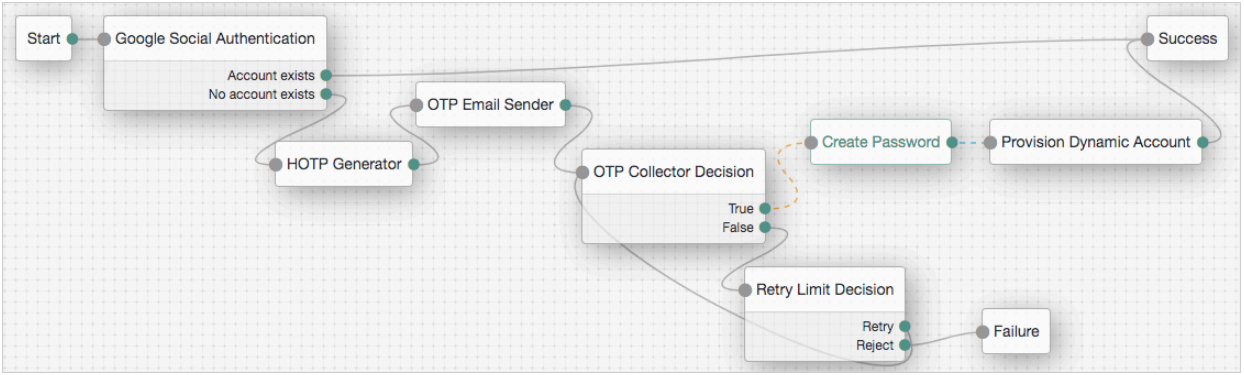


Figure 57. Google-DynamicAccountCreation Tree With Create Password node

Consent Collector node

The Consent Collector node prompts the user to consent to share their profile data.

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

A consent notice is listed for each IDM mapping that has consent enabled. If an IDM mapping is not created, or the mappings do not have privacy and consent enabled, no consent message will be shown to the user.

This node is primarily used in progressive profile or registration flows.

Properties

Property	Usage
All Mappings Required	If enabled, all mappings listed by this node require consent in order to move forward.
Privacy & Consent Message	Localized message providing the privacy and consent notice. The key is the language (such as `en` or `fr`), and the value is the message to display.

Display Username node

This node is used to fetch a username based on a different identifying attribute (such as an email address), then display it on screen. To email the username to the user instead, use the Identify Existing User node combined with a Email Suspend node or Email Template node. The Display Username node requires IDM integration to function.

Properties

Property	Usage
User Name	The attribute used to identify the username in an IDM object.
Identity Attribute	The attribute used to identify the object in IDM. Since this node is generally used for recovering a username, the identity attribute in this case should be some other attribute that is unique to a user object, such as the user's email address. You will receive an exception if there is more than one result for this attribute, so make sure the value of whatever attribute you select is unique for each user.

Identify Existing User node

This node verifies a user exists based on an identifying attribute, such as an email address, then makes the value of a specified attribute available in a tree's shared state.

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

For example, use this node in a "Forgot Username" flow to fetch a username to email to the user. If you want to display the username on screen, use the Display Username node instead.

Properties

Property	Usage
Identifier	The attribute to collect from an IDM object.
Identity Attribute	The attribute used to identify the object in IDM. Since this node is generally used for recovering a username, the identity attribute in this case should be some other attribute that is unique to a user object, such as the user's email address.

Example

The following is an example of a forgotten password tree. The user enters information that the Identify Existing User will use to try to identify them. Next, AM uses the Email Suspend node to send an email to the user and suspend the authentication tree. Once authentication is resumed, the user is sent to a different tree to reset their password:

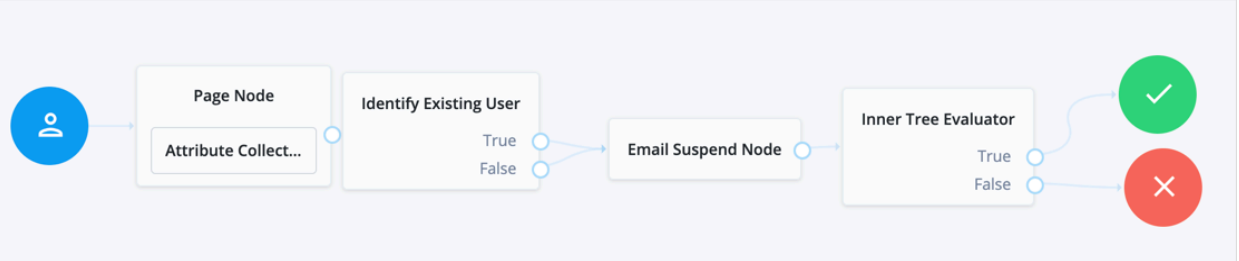


Figure 58. Identify Existing User Tree

KBA Decision node

The KBA Decision node is used to check if the minimum number of KBA questions required by the system are defined for the user.

NOTE

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

The number of KBA questions is determined by the `minimumAnswersToDefine` property in `selfservice.kba.json` in IDM. This node is mainly used for Progressive Profile completion.

Properties

Property	Usage
Identity Attribute	The attribute used to identify the object in IDM.

KBA Definition node

The KBA Definition node collects KBA questions and answers from the user and saves them to the user object.

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

This is used when creating or updating a user with Knowledge-Based Authentication enabled. For more information, see Security Questions.

Properties

Property	Usage
Purpose Message	A localised message describing the purpose of the data requested from the user.

KBA Verification node

The KBA Verification node presents KBA questions to the user, collects answers to those questions, and verifies the input against the user's stored answers.

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

This is used during self-service actions such as Forgot Password or Forgot Username, where additional authentication is needed. The number of KBA questions is determined by the `minimumAnswersToVerify` property in `selfservice.kba.json` in IDM.

Properties

Property	Usage
KBA Attribute	The IDM object attribute in which KBA questions and answers are stored.
Identity Attribute	The attribute used to identify the object in IDM.

Pass-Through Authentication node

The Pass-Through Authentication node authenticates an identity through a connector to a third-party service. This lets you migrate user profiles without forcing users to reset their passwords, or retain a third-party service indefinitely as the canonical store for authentication credentials.

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

Before you use the node:

- Configure the connector to the third-party service.

  For details, see Connectors in the IDM documentation.

- If you plan to collect credentials in the identity repository for users, synchronize accounts from the third-party service.

  For details, see Synchronization in the IDM documentation.

Use this node after collecting the authentication credentials. For example, use the Platform Username node and Platform Password node to collect the username and password.

Pass the credentials to this node to authenticate the identity against the service.

Properties

Property	Usage
System Endpoint	Required. Name of the connector to the third-party service that performs authentication.
Object Type	The OpenICF object type for the object being authenticated. Default: `account`
Identity Attribute	The username attribute for authentication. Default: `userName`
Password Attribute	The password attribute for authentication. Default: `password`

Example

Before trying this example, synchronize accounts from the third-party service. The example shows a login journey that tries pass-through authentication when local authentication fails, and stores the user password when authentication with the third-party service succeeds.

In this example, the user enters their credentials with the Platform Username node and Platform Password node. The Data Store Decision node authenticates against the platform directory service. On failure, authentication passes through to the third-party service. If authentication with the third-party service is successful, the Identify Existing User node and Required Attributes Present node check for a valid user profile. The Patch Object node updates the user's profile with the successful password:

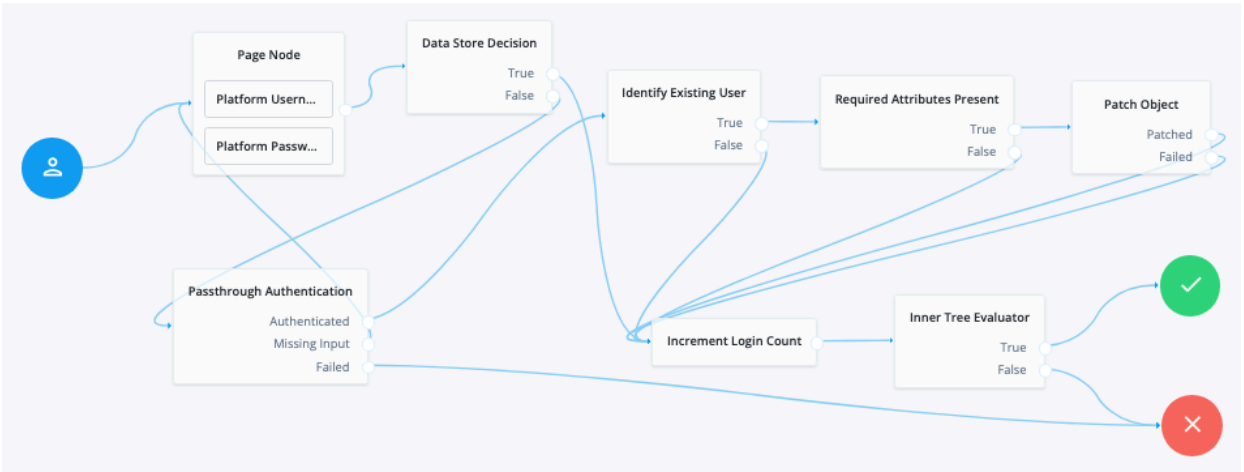


Figure 59. Pass-Through Authentication

Patch Object node

The Patch Object node is used to update attributes in an existing managed object in IDM.

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

This is used in cases such as progressive profile completion, where you may wish to collect additional profile data from a user after they have logged in several times.

Properties

Property	Usage
Patch as Object	Allows patching as the object being updated. Enable this property to patch a user object as part of the user's current session, such as when updating their password.
Ignored Fields	Fields from the tree's shared state that should be ignored as part of patch. If this is empty, all shared-state fields in the tree's `nodeState` object are attempted as part of the patch. Use this to keep your patch focused only on the fields you want to update.
Identity Resource	The type of IDM managed identity resource object that this node will create. It must match the identity resource type for the current tree. TIP To check for the available managed identity resource types, go to the IDM admin UI, and open the **Manage** drop-down list, at the upper right corner of the screen. Identity managed object types are preceded by the 👤 icon.

Property	Usage
Identity Attribute	The attribute used to identify the object to update in IDM.

Platform Password node

This node prompts the user to enter their password and stores the input in a configurable state attribute.

Use this node instead of the Password Collector node when you have configured AM as part of a ForgeRock Identity Platform deployment.

IMPORTANT

This node uses the `_id` of the object for policy evaluation. For existing users, the user's `_id` must therefore be in the shared state for user-specific policies (such as password history, cannot-contain-others, and so on) to be evaluated. (No `_id` is available for new users.)

Properties

Property	Usage
Validate Password	When enabled, this node checks the user's input against IDM's password policies, and returns any policy failures as errors. For example, if you submitted an invalid password on registration, the response from this node would include a list of failed policies: `{ "name": "failedPolicies", "value": [ "{ \"params\": { \"minLength\": 8 }, \"policyRequirement\": \"MIN_LENGTH\" }", "{ \"params\": { \"numCaps\": 1 }, \"policyRequirement\": \"AT_LEAST_X_CAPITAL_LETTERS\" }", "{ \"params\": { \"numNums\": 1 }, \"policyRequirement\": \"AT_LEAST_X_NUMBERS\" }" ] },`
Password Attribute	The attribute used to store a password in the IDM object.

Platform Username node

This node prompts the user to enter their username, and stores it in a configurable state attribute.

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

Use this node instead of the Username Collector node when working with AM and IDM as an integrated platform.

Properties

Property	Usage
Validate Username	When enabled, this node checks the user's input against IDM's username policies, and returns any policy failures as errors.
Username Attribute	The attribute used to store a username in the IDM object.

Profile Completeness Decision node

The Profile Completeness Decision node is used in progressive profile flows. It checks how much of a user's profile has been filled out, where the completeness of a profile is expressed as a percentage of user-viewable, user-editable fields that are not `null`.

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

Properties

Property	Usage
Profile Completeness Threshold	Percentage of user-viewable and user-editable fields in a profile that need to be filled out for the node to pass. Expressed as a number between 0 and 100.
Identity Attribute	The attribute used to identify the object in IDM.

Query Filter Decision node

Checks if the contents of a user's profile matches a specified query filter.

NOTE

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

Use this node to verify whether a particular field has been filled out, or that the contents of a field match a specific pattern. For instance, use this in progressive profile flows to check if marketing preferences are set on a user's profile.

For more information on constructing effective query filters, see Construct Queries in the IDM documentation.

Properties

Property	Usage
Query Filter	A query filter used to check the contents of an object.
Identity Attribute	The attribute used to identify the object that will be queried in IDM.

Required Attributes Present node

The Required Attributes Present node checks the specified identity resource in IDM (by default, `managed/user`), and determines if all attributes required to create the specified object exist within shared state of the tree.

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

Properties

Property	Usage
Identity Resource	The type of IDM managed identity resource object that this node will create. It must match the identity resource type for the current tree. TIP To check for the available managed identity resource types, go to the IDM admin UI, and open the **Manage** drop-down list, at the upper right corner of the screen. Identity managed object types are preceded by the icon.

Select Identity Provider node

This node is used in combination with the Social Provider Handler node to enable use of the Social Identity Provider Service. It presents the user with a list of configured, enabled, social identity providers to use for authentication.

The node has two possible outputs: social authentication, and local authentication. Local authentication can be turned off by disabling **Include local authentication**.

This node returns the SelectIdPCallback when more than one social identity provider is enabled, or a single provider is enabled as well as the **Local Authentication** option, and therefore a choice from the user is required. If no choice from the user is required, authentication proceeds to the next node in the tree.

Properties

Property	Usage
Include local authentication	Determines whether local authentication will be included as an available method for authenticating.
Offer only existing providers	ForgeRock Identity Platform deployments only.
Password attribute	ForgeRock Identity Platform deployments only.

Property	Usage
Identity Attribute	ForgeRock Identity Platform deployments only.
Filter Enabled Providers	By default, the node displays all identity providers that are marked as **Enabled** in the Social Identity Provider Service as a selectable option. Specify the name of one of more providers to filter the list. TIP View the names of your configured social identity providers by navigating to Services > Social Identity Provider Service > Secondary Configurations. If this field is not empty, providers must be in the list, and also be enabled in the Social Identity Provider service, in order to be displayed. If left blank, all enabled providers are displayed.

Terms and Conditions Decision node

The Terms and Conditions Decision node verifies the user has accepted the active set of Terms and Conditions.

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

You set up Terms and Conditions in the Platform UI. For more information, see Terms & Conditions in the platform documentation.

Use this node when you want to verify the user has accepted your Terms and Conditions before proceeding (such as logging in, or in a progressive profile tree). This is often used with the Accept Terms and Conditions node.

Properties

Property	Usage
Identity Attribute	The attribute used to identify the object to check in IDM.

Time Since Decision node

Checks if a specified amount of time has passed since the user was registered.

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

For example, if you wanted to prompt users to review your terms and conditions after the account is a week old, you could set the `Elapsed Time` property to `10080` minutes. After that time has elapsed, the next time the user logs in, they will be prompted to review your terms and conditions.

This node is mainly used for Progressive Profile completion.

Properties

Property	Usage
Elapsed Time	The amount of time since the user was created, in minutes, that needs to elapse before this node is triggered. This property also supports specifying basic time units. For example, when setting the property to `10080` minutes, writing `7 days` or `1 week` also works.
Identity Attribute	The attribute used to identify the object to update in IDM.

Utility authentication nodes

Use the following nodes to perform various tasks during the authentication flow:

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.

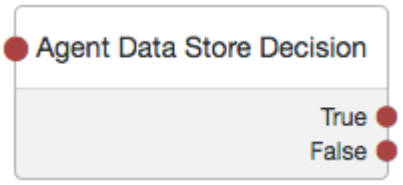
NOTE

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.

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.

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.

Property	Usage
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.

Email Suspend node

The Email Suspend node is used to generate and send an email to a user, such as an address verification email, based on an email template in IDM. The authentication tree will pause until the user clicks a link in the email to resume the tree flow.

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

The link is generated by the Email Suspend node, and is passed along to IDM as part of the email object, in a property called `resumeURI`.

This node uses the email service configured in IDM to send email. If you do not need the auth tree to pause and wait for a response from email, use the Email Template node instead.

Properties

Property	Usage
Email Template Name	The name of the IDM email template to be sent. Check IDM for the names of available email templates, or to create a new template.
Email Attribute	The IDM attribute storing the address to send the email to.
Email Suspend Message	The localized message to be returned once the tree is suspended. The default message is, "An email has been sent to your inbox."

Property	Usage
Object Lookup	Determines whether the object should be looked up in IDM. If true, IDM is queried for an existing object. Otherwise, the object in the authentication tree's shared state is used. For example, if suspending a user registration flow before the user object is created in IDM, this should be set to `false`. If the registration flow has already created the new user object when the flow is suspended, then this should be set to `true`.
Identity Attribute	The attribute used to identify the object in IDM.

Example

The following is an example of a forgotten password tree. The user enters information that the Identify Existing User node will use to try to identify them. Next, AM uses the Email Suspend node to send an email to the user and suspend the authentication tree. Once authentication is resumed, the user is sent to a different tree to reset their password:

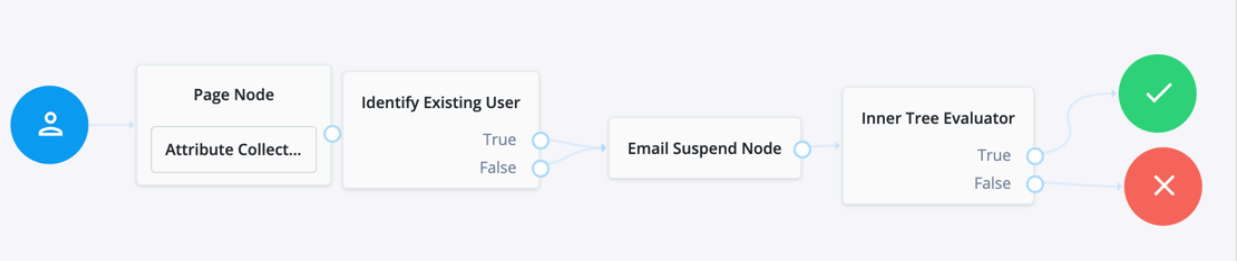


Figure 60. Email Suspend Tree

Email Template node

The Email Template node is used to generate and send an email to a user, such as a welcome email, based on an email template in IDM.

NOTE

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

This node uses the email service configured in IDM to send email. If you need the auth tree to pause and wait for a response from email, use the Email Suspend node instead.

This node has two possible outcomes: "Email Sent" and "Email Not Sent", which can be used if you need different behavior depending on the outcome. According to OWASP authentication recommendations, the message to the user should be the same in both cases.

Properties

Property	Usage
Email Template Name	The name of the IDM email template to be sent. Check IDM for the names of available email templates, or to create a new template.
Email Attribute	The IDM attribute storing the address to send the email to.
Identity Attribute	The attribute used to identify the object in IDM.

Failure URL node

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

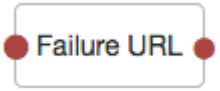
NOTE

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 Success and failure redirection URLs.

TIP

The URL is also saved in the tree's `nodeState` object for the key, `failureUrl`, which can be useful for custom node developers. For more information, see Customizing Authentication Trees.



Properties

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

Get Session Data node

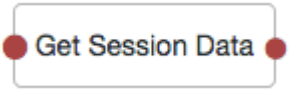
The Get Session Data authentication node retrieves the value of a specified key from a user's session data, and stores it in the specified key in the tree's `nodeState` object.

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";
}
```



Properties

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

Example

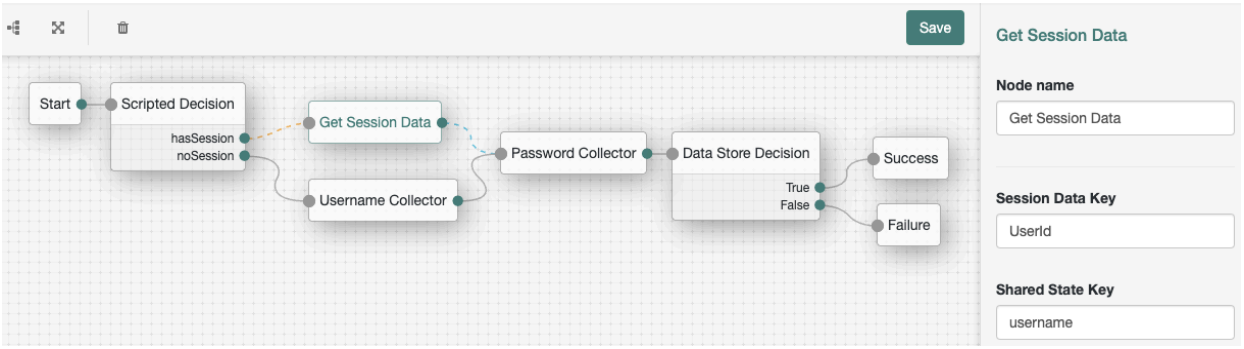


Figure 61. Get Session Data Tree Example (Standalone AM)

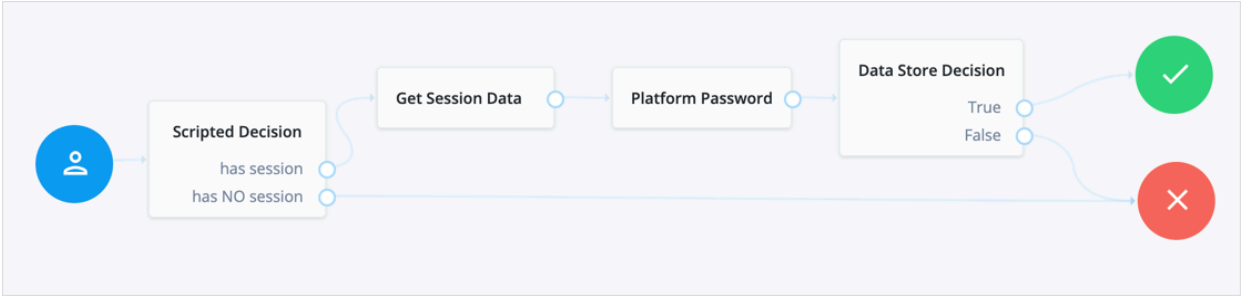


Figure 62. Get Session Data Tree Example (ForgeRock Identity Platform)

The following table includes example keys that may be available in an existing session, and corresponding sample values:

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`
`AuthLevel`	`0`
`CharSet`	`UTF-8`
`clientType`	`genericHTML`
`FullLoginURL`	`/openam/XUI/?realm=alpha#login/`
`Host`	`198.51.100.1`
`HostName`	`openam.example.com`
`Locale`	`en_US`

Key	Sample value
`Organization`	`dc=openam,dc=forgerock,dc=org`
`Principal`	`uid=amAdmin,ou=People,dc=openam,dc=forgerock,dc=org`
`Principals`	`amAdmin`
`Service`	`ldapService`
`successURL`	`/openam/console`
`sun.am.UniversalIdentifier`	`uid=amAdmin,ou=People,dc=openam,dc=forgerock,dc=org`
`UserId`	`amAdmin`
`UserProfile`	`Required`
`UserToken`	`amAdmin`
`webhooks`	`myWebHook`

Inner Tree Evaluator node

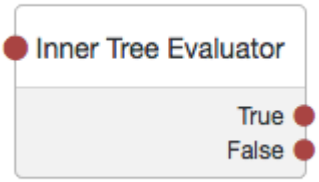
The Inner Tree Evaluator authentication node allows the nesting and evaluation of authentication trees as children within a parent tree. There is no limit to the depth of nested trees.

Any information collected or set by the parent tree, such as a username or the authentication level, is available to child trees.

Shared node state data collected by child trees is available to the parent when evaluation of the child is complete, but data stored in transient and secure state is not. For instance, if a child tree collects and stores the user's password in transient state, it cannot be retrieved by a node in the parent tree when evaluation continues.

For information about shared state data, refer to Access shared state data.

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.



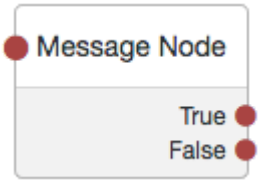
Properties

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

Message node

The Message authentication node lets you 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.

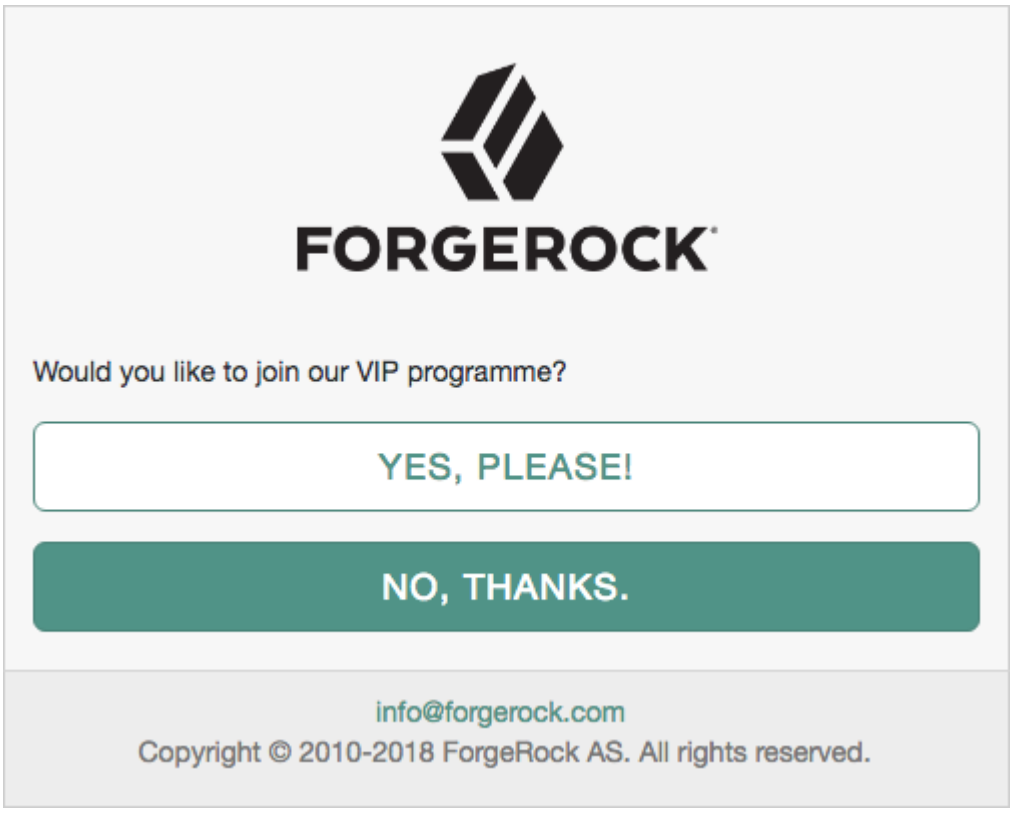


Properties

Property	Usage
Message	Click **Add**. Enter the message locale in the `Key` field; for example `en-gb`. Enter 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 (🗑).

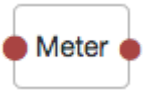
Property	Usage
Positive answer	Specify a positive answer that will cause tree evaluation to continue along the `True` outcome path. Click the Add button, and then enter the locale of the positive answer in the `Key` field, and the message to display to the user in the `Value` field. If the locale of 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



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. The metric is exposed in all available interfaces, as described in Monitor AM instances.



Properties

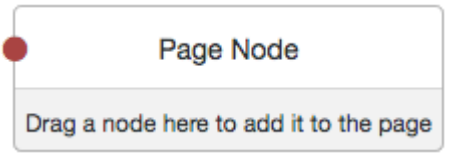
Property	Usage
Metric Key	Specify the name of a metric to increment when tree evaluation passes through the node.

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

Property	Usage
Header	Optional. Localized title for the page node and the nodes contained within it. Use this when components of an authentication flow need a title, such as breaking a registration into labeled sections.
Description	Optional. A localized description for the page node and the nodes contained within it. Use this when additional descriptive text is needed in an authentication flow.
Stage	Optional. This is used in UI development, to help identify what node or series of nodes are being returned so they can be rendered in the UI appropriately.

NOTE

The Page node's optional properties are passed in the response, but the UI needs to support these properties before they will be visible to the end user.

Example

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

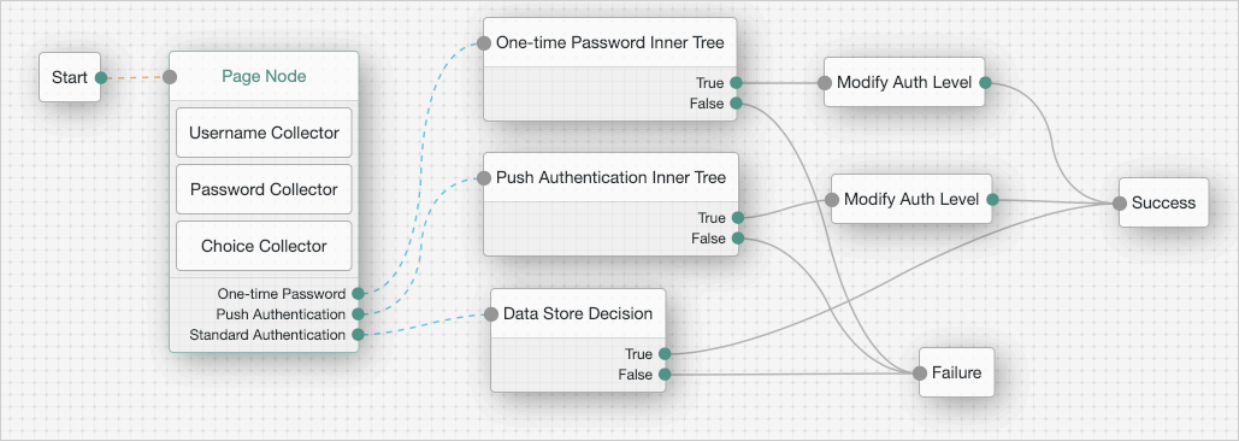


Figure 63. Example Tree With Page node (Standalone AM)

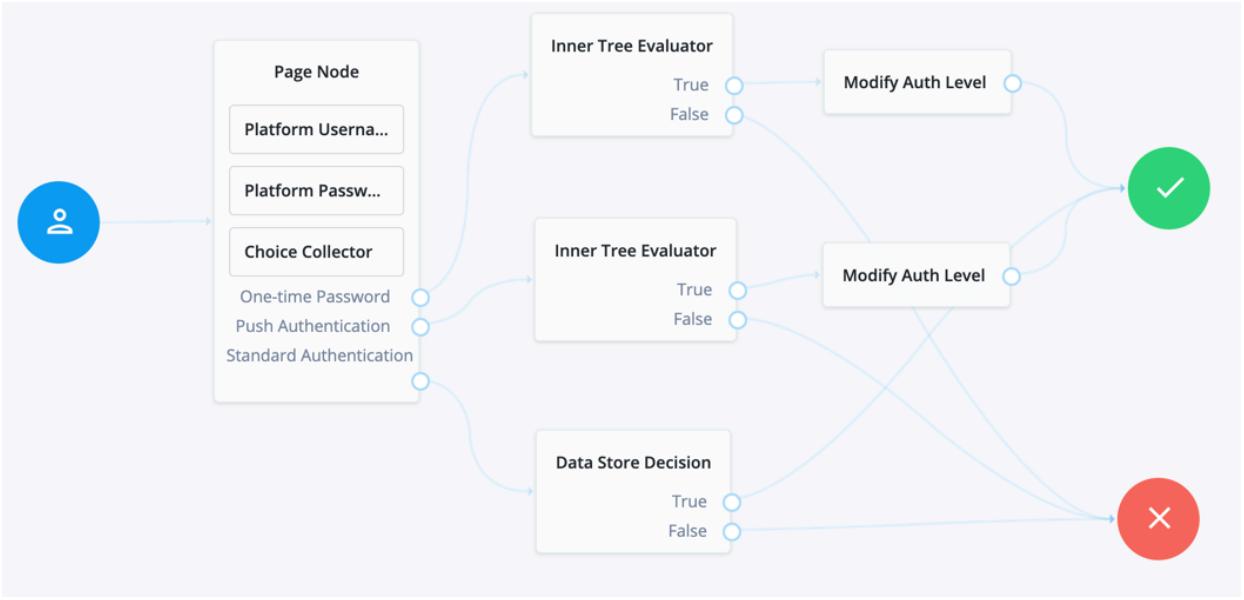


Figure 64. Example Tree With Page node (ForgeRock Identity Platform)

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

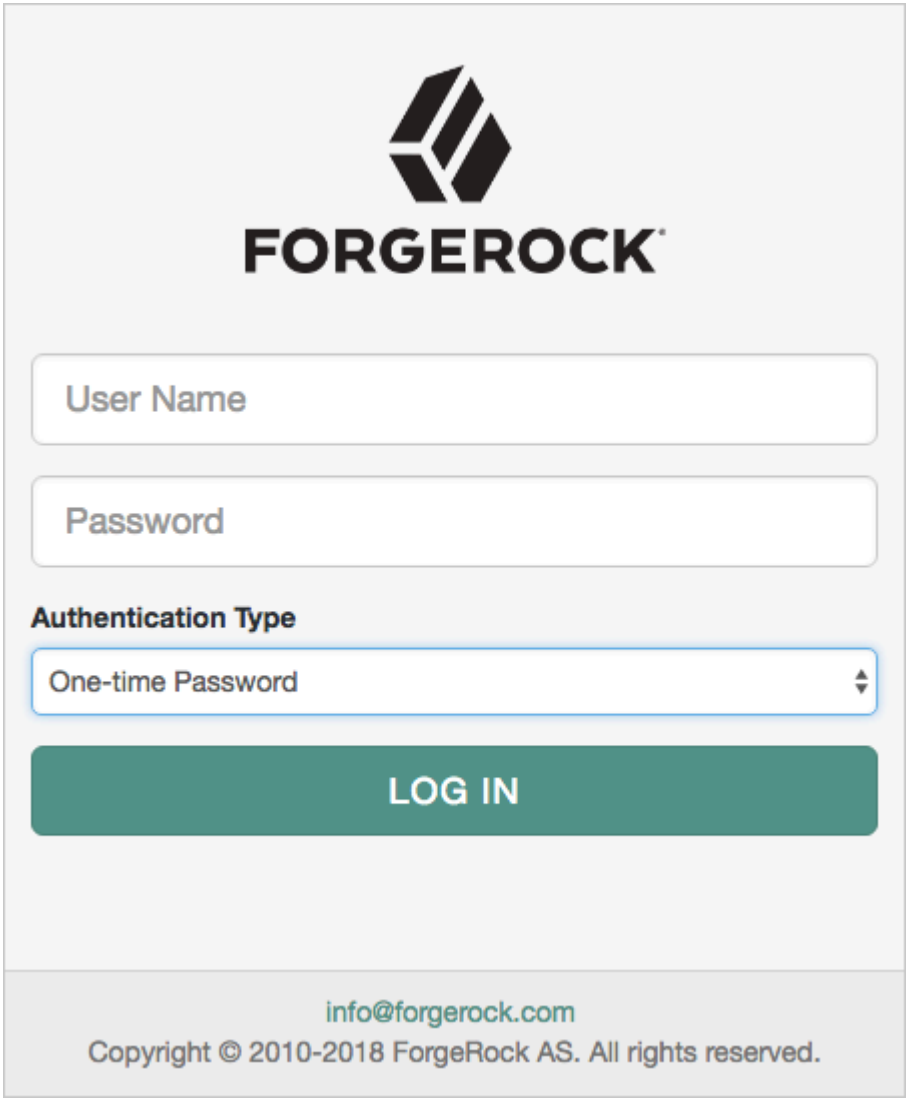


Figure 65. User View of Example Tree with Page node (Standalone AM)

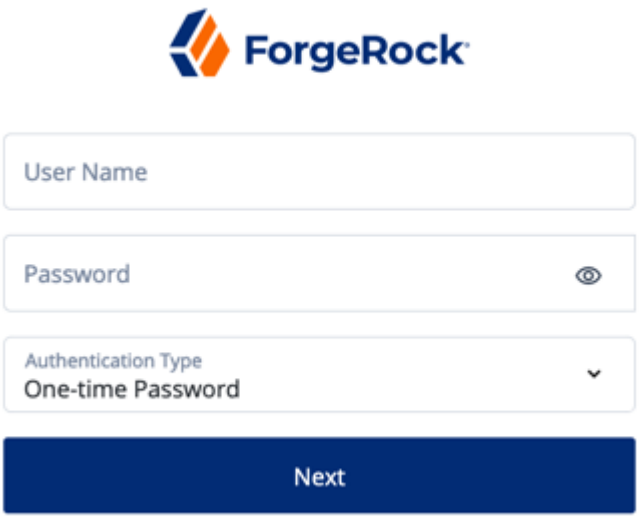


Figure 66. User View of Example Tree with Page node (ForgeRock Identity Platform)

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 \
--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/realms/alpha/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"
```

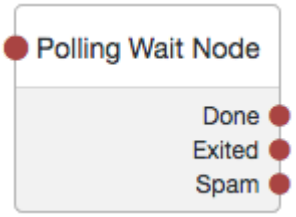
For more information on authenticating using the REST API, see Authenticate over REST.

When using the UI for authentication, it automatically waits for the required amount of time and resubmits 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

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. 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 11 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.

Property	Usage
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 11 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.

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

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

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

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: • 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. • Disabled. 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. Default: Enabled.

Example

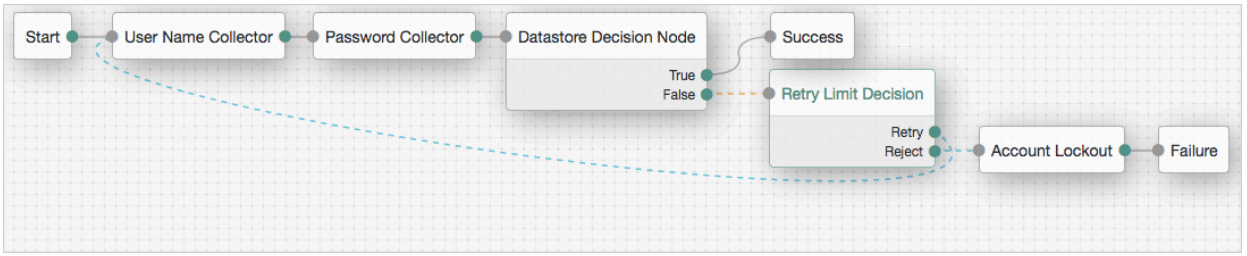


Figure 67. RetryLimit Tree Example (Standalone AM)

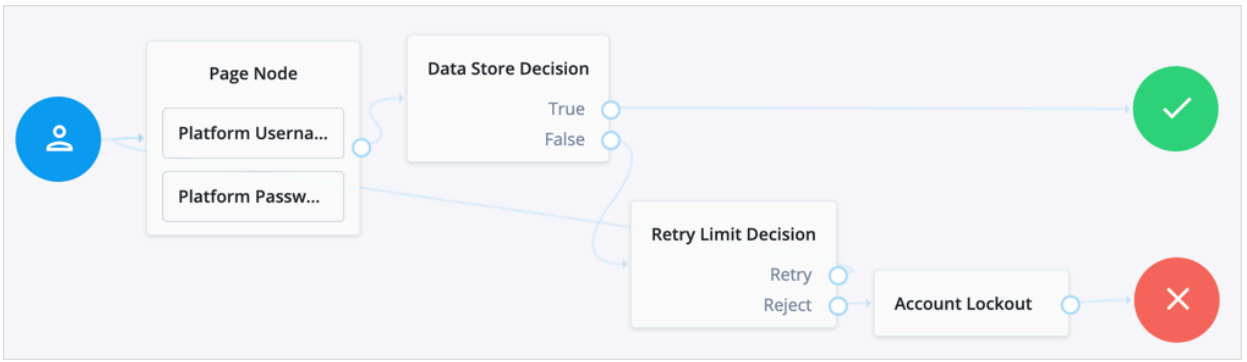


Figure 68. RetryLimit Tree Example (ForgeRock Identity Platform)

Scripted Decision node

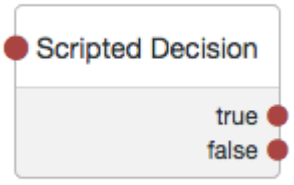
The Scripted Decision authentication node allows execution of scripts during authentication. Tree evaluation continues along the path matching the result.

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 Manage scripts (UI).

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

All of the inputs required by the script and the outputs produced by it must be declared in the node's configuration or the script may fail. Even if the definition is `null`, it still needs to be declared. Use the wildcard `*` to include any available inputs or outputs.

For information about the API available for use in the Scripted Decision node, see Scripted decision node API.



Properties

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.
Script Inputs	A list of state inputs required by the script. Defaults to `*`.
Script Outputs	A list of state outputs produced by the script. Defaults to `*`.

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.

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

Properties

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.

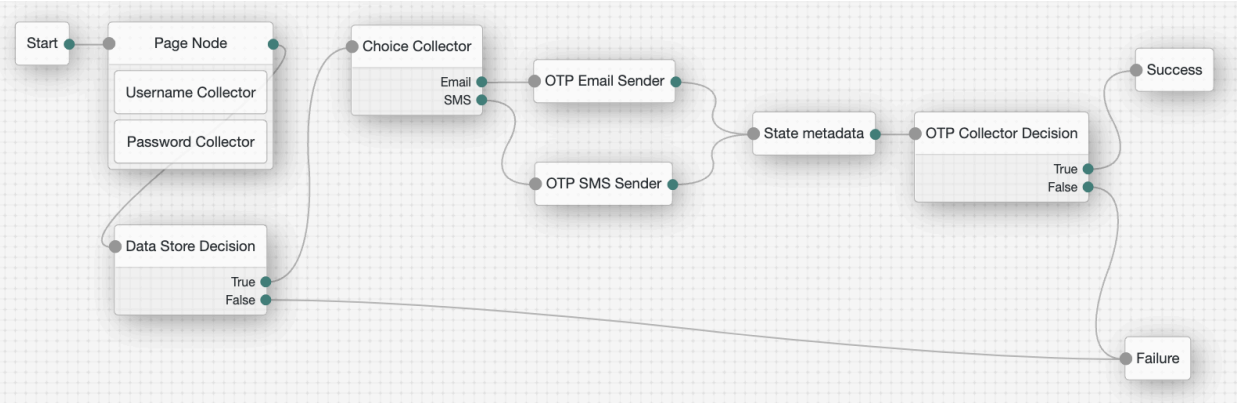


Figure 69. Example tree with the State Metadata node (Standalone AM)

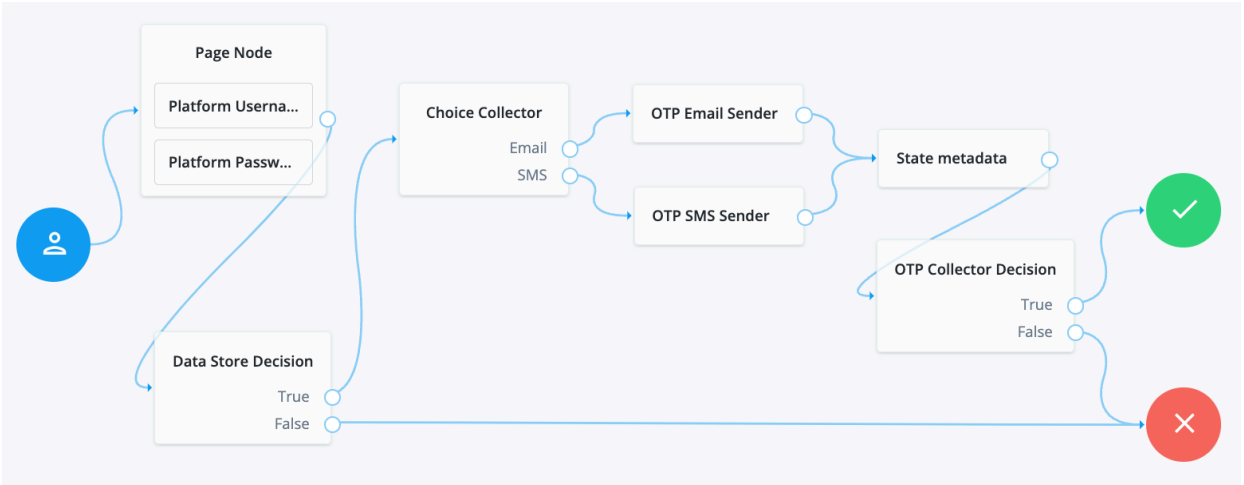


Figure 70. Example tree with the State Metadata node (ForgeRock Identity Platform)

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.

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 Success and failure redirection URLs.

TIP

The URL is also saved in the tree's `nodeState` object, for the key, `successUrl`, which can be useful for custom node developers. For more information, see Customizing Authentication Trees.

Properties

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

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.

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 Monitor AM instances.

Properties

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.

Thing authentication nodes

Use the following nodes to perform various tasks related to authenticating IoT things:

Authenticate Thing node

This node authenticates a *thing*. A thing represents an IoT device, service, or the IoT Gateway. Before you configure this node, ensure that the IoT Service is configured for the realm.

IMPORTANT

Support for this node is provided by the IoT SDK.

The node supports two methods of authentication:

- Proof of Possession JWT

  The node collects a proof-of-possession JWT from the request and does the following:

  - Checks that the claims are valid.
  - Checks that an identity with the same ID as the name of the JWT subject exists.
  - Checks that the identity contains a confirmation key that matches the JWT's `kid`.
  - Validates the JWT's signature, using the confirmation key stored in the identity.
- Client Assertion

  The node collects a JWT Bearer token from the request for authentication and validates the request according to the JWT Profile for OAuth 2.0 Client Authentication and Authorization Grants.

If all checks are successful, the tree continues through the `Success` path, and adds the username and the verified claims to the authentication tree's shared state.

If the identity does not exist, or AM cannot match the identity with the confirmation key, the tree continues through the `Requires Registration` outcome. If any other check

fails, the tree continues through the `Failure` outcome.

Properties

Property	Usage
JWT Authentication Method	Choose the required JWT authentication method. Options are as follows: • **Proof of Possession** Prove that the signer of the JWT is the owner of the key by including a challenge nonce in the JWT. Validation is done in accordance with the JWT Proof of Possession specification. • **Client Assertion** Present a JWT Bearer token for authentication, and validates the request according to the JWT Profile for OAuth 2.0 Client Authentication and Authorization Grants.
Issue Restricted Token	If this setting is enabled, a Proof of Possession restriction is added to the session token issued on successful authentication. Any requests accompanied by the token must be signed with the key that was used to sign the authentication JWT.
Additional Audience Values	Specify any additional audience values that will be permitted when verifying JWTs. These audience values are in addition to the AM base, issuer and token endpoint URIs for the Client Assertion authentication method or the realm path for Proof of Possession.

Examples

The following example shows how to authenticate a thing when the identity already exists in the identity store and when its profile contains a confirmation key:

Figure 71. Authenticating a Thing Without Registration

The following example shows how to authenticate a thing when the identity does not exist, or when it needs to refresh its confirmation key:

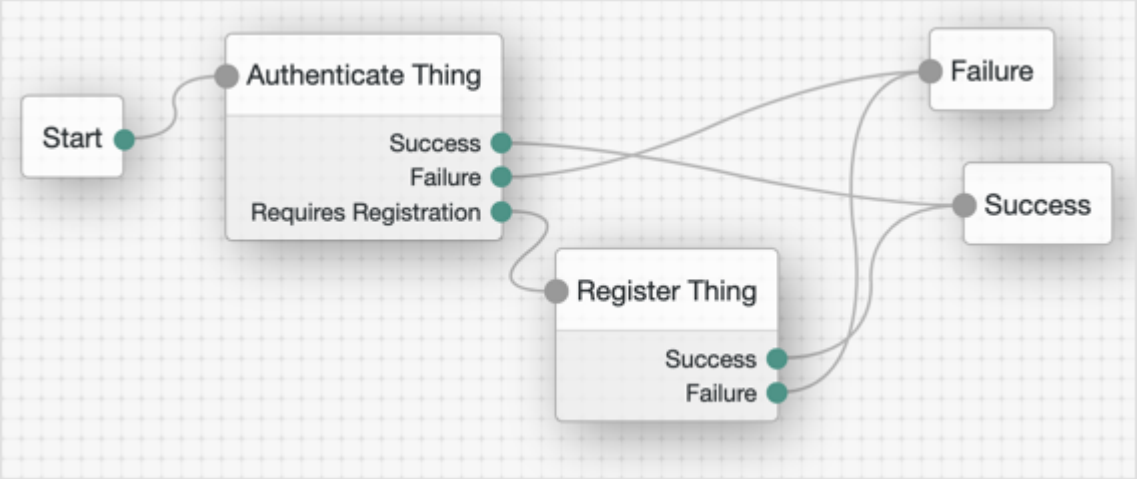


Figure 72. Authenticating a Thing With Registration

Register Thing node

This node registers a thing. A thing represents an IoT device, service, or the IoT Gateway. Before you use this node, ensure that the IoT Service is configured for the realm.

IMPORTANT

Support for this node is provided by the IoT SDK.

The node collects a JWT from the request, and validates the JWT according to the configured JWT registration method.

If the JWT is valid, the node uses the claims in the JWT to create an identity for the thing and register (or rotate) a confirmation key for it. Then, the tree continues through the `Success` outcome.

If the node cannot validate the JWT, the tree continues through the `Failure` outcome.

For an example on how to use this node, see Authenticate Thing node.

Properties

Property	Usage
JWT Registration Method	Choose the method that the node will use to validate the JWT: • **Proof of Possession & Certificate** Register using a Proof of Possession JWT that includes an X.509 certificate for providing trust. A challenge nonce is presented in the callback, and must be included in the signed JWT. • **Proof of Possession & Software Statement** Register using a Proof of Possession JWT and a Software Statement for providing trust. A challenge nonce is presented in the callback, and must be included in the signed Proof of Possession JWT. The claims in the Software Statement take precedence over the claims in the Proof of Possession JWT. • **Proof of Possession** Register using a Proof of Possession JWT without using a trusted third party. A challenge nonce is presented in the callback, and must be included in the signed JWT. • **Software Statement** Register using a Software Statement, without doing proof of possession. If you select this registration method, the resultant session token will not include a proof of possession restriction. Default: Proof of Possession & Certificate

Property	Usage
Verify Certificate Subject	If the configured JWT registration method is `Proof of Possession & Certificate`, this option verifies that the subject provided in the JWT is the same as the X.509 certificate subject CN or UID. Default: Enabled
Create Identity	Specifies whether AM will create an ID for the thing if one does not exist. Default: Disabled
Rotate Confirmation Key	Specifies whether multiple confirmation keys can be registered for a thing. Disable this setting to allow only one key per thing. Default: Disabled
Default Attribute Values	Lets you set default values for the thing's attributes, where **KEY** is the name of the attribute in the data store, and **VALUE** is the default value of the attribute.
Claim to Attribute Mapping	If **Create Identity** is enabled, this property lets you map verified claims in the JWT to attributes in the thing identity. **KEY** is the claim name and **VALUE** is the name of the attribute in the data store.
Overwrite Attributes	Specifies whether the node will overwrite the value for an existing profile attribute when a claim with a different value is provided in the JWT. Default: Disabled

Scripted decision node API

The scripted decision node lets you write a server-side script, in JavaScript or Groovy, to determine the path the authentication journey takes.

These scripts have access to a number of *bindings*, which provide the context to help you make the decision.

The primary role of a scripted decision node is to specify the possible paths a user can take. There are two methods to define these paths within the script:

outcome

The simplest method is to assign one or more string values to the `outcome` variable.

```
if (...) {
  outcome = "true"
} else {
  outcome = "false"
}
```

When configuring the scripted decision node in an authentication tree, add the two outcomes `true` and `false`, and connect them to other parts of the tree, so that tree evaluation can continue.

You can specify as many outcomes as required in your scripts; for example, you might have `hours`, `days`, and `months`. Be sure to specify each possible outcome when designing your authentication journey.

Action

You can use the `Action` interface to define the script outcome and/or specify an operation to perform, for example, by chaining with `ActionBuilder` functions.

▼ `Action.ActionBuilder` functions:

Method	Information
`public ActionBuilder putSessionProperty(String key, String value)`	Add a session property. Refer to Setting session properties.
`public ActionBuilder removeSessionProperty(String key)`	Remove an existing session property.
`public ActionBuilder withDescription(String description)`	Set a description for the action.
`public ActionBuilder withErrorMessage(String message)`	Set an error message to display to the end user when the journey reaches the Failure node.

Method	Information

`public ActionBuilder withIdentifiedIdentity(AMIdentity id)`
 	
`public ActionBuilder withIdentifiedIdentity(String username, IdType id)`
	Set the identity, authenticated or not, of the user or agent verified to exist in an identity store.
	
	Use these methods to record the type of identified user. If the advanced server property, `org.forgerock.am.auth.trees.authenticate.identified.identity` is set to true, AM uses the stored identified identities to decide who to log in.
	
	This lets the authentication tree engine correctly resolve identities that have the same username.
	
	For more information, refer to advanced server properties.

`public ActionBuilder withLockoutMessage(String message)`	Set an error message to display to the end user when the account is locked or inactive.
`public ActionBuilder withStage(String stage)`	Set a stage name to return to the client to aid the rendering of the UI. The property is only sent if the script also sends callbacks.

For the full API, refer to ActionBuilder.

Example:

```
var fr = JavaImporter(
    org.forgerock.openam.auth.node.api.Action
)

if (...) {
  // Set outcome to "true", and create and populate a custom
session property:
  action =
fr.Action.goTo("true").putSessionProperty("customKey",
"customValue").build()
}
```

```
else
{
  // Set outcome to "false". If supported by the UI, the error
message is displayed:
  action = fr.Action.goTo("false").withErrorMessage("Friendly
error description.").build()
}
```

TIP

You can also use the `Action` interface for other functionality:

- Setting session properties
- Using Callbacks

For more information on the `Action` interface, refer to Action in the *AM Javadoc*.

NOTE

An outcome specified as an `Action` takes precedence over the value set for the `outcome` variable:

```
action = Action.goTo("false").build() // Tree continues along "false"
outcome.

outcome = "true" // No effect.
```

For more information on specifying outcomes when using the scripted decision node, refer to Scripted Decision node.

The following table lists the bindings accessible to scripted decision node scripts:

Scripted decision node bindings

Binding	Information
`auditEntryDetail`	Add information to the AM audit logs. Refer to Adding audit information.
`callbacks`	Request additional data from the user, by sending any of the supported callbacks. Refer to Using Callbacks.

Binding	Information
`existingSession`	If the user has previously authenticated and has a session, use this variable to access the properties of that session. The user will only have an existing session when performing a session upgrade. Any properties that may have been added by nodes earlier in the current tree will not appear on the user's new session until the authentication tree is completed, and are therefore not available to the `existingSession` variable. Refer to Accessing existing session properties.
`httpClient`	Make outbound HTTP calls. Refer to Accessing HTTP Services.
`idRepository`	Access the data stored in the user's profile. Refer to Accessing profile data.
`logger`	Write information to the AM debug logs. Refer to Debug logging.
`realm`	Return the name of the realm to which the user is authenticating as a string. For example, authenticating to the top-level realm returns a string value of `/` (forward-slash). Authenticating to a subrealm of the top level realm might return `/subRealm`.
`requestHeaders`	Access the HTTP headers provided in the login request. Refer to Accessing request header data.

Binding	Information
`requestParameters`	Access the HTTP request parameters provided in the login request. Refer to Accessing request parameter data.
`secrets`	Access the secrets configured in an AM instance. Refer to Accessing credentials and secrets.
`nodeState`	Access data set by previous nodes in the tree, or store data to be used by subsequent nodes. Refer to Accessing shared state data.

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.

Methods

`String[] requestHeaders.get(String HeaderName)`

Return a java.util.ArrayList of the values in the named request header, or `null`, if the property is not set. Note that header names are case-sensitive.

Example:

```
var headerName = "user-agent"

if (requestHeaders.get(headerName).get(0).indexOf("Chrome") !==
-1) {
    outcome = "true"
} else {
    outcome = "false"
}
```

Accessing request parameter data

Scripted decision node scripts can access any query parameters provided by the login request by using the methods of the `requestParameters` object.

Note that the script has access to a copy of the parameters. Changing their values does not affect the request itself.

Methods

`String[] requestParameters.get(String ParameterName)`

Return a java.util.ArrayList of the values in the named request parameter, or `null`, if the parameter is not available.

Example:

JavaScript | Groovy

```
var service
var authIndexType = requestParameters.get("authIndexType")

if (authIndexType && String(authIndexType.get(0)) ===
"service") {
    service =
requestParameters.get("authIndexValue").get(0)
}
```

NOTE

In JavaScript, the values stored in `requestParameters` have a `typeof` of *object*, and represent the `java.lang.String` class. Convert the value to a string in order to use strict equality comparisons.

Accessing shared state data

A script can access the shared state of the tree with the methods of the `nodeState` object.

There are three types of state:

Shared

Non-sensitive state.

Transient

Sensitive state.

Transient state data is never sent back to the user's browser in a callback so doesn't need to be encrypted. The transient state is for sensitive data and for data not required after the next callback.

Secure

Encrypted sensitive state.

Secure state data is sent back to the user's browser encrypted as part of the shared state object.

Transient state data is *promoted* to secure state data when:

- A callback to the user is about to occur.
- A downstream node is detected in the journey, requesting data in the transient state as script input.

IMPORTANT

Unless the downstream node explicitly requests the secure state data by name, the authentication journey removes it from the node state after processing the next callback.

For example, a node in a registration journey stores a user's password in transient state. The node sends a callback to the user before an inner tree node, downstream in the journey, consumes that password. As part of the callback, the journey assesses what to add to the secure state. It does this by checking the state inputs that downstream nodes in the journey require. Nodes that *only* request `*` are ignored, as this would result in putting everything that's in transient state into secure state, and retaining sensitive information longer than necessary.

If a downstream node requires the password, it must therefore explicitly request it as state input, even if it lists the `*` wildcard as input.

Methods

***`JsonValue nodeState.get(`String Property Name)` ***

Returns the value of the named property. The value may come from the transient, secure, or shared states, in that order. For example, if the same property is available in several states, the method will return the value of the property in the transient state first.

If the property is not set, the method returns `null`.

Note that property names are case-sensitive.

Example:

```
var currentAuthLevel = nodeState.get("authLevel").asString()
var givenPassword = nodeState.get("password").asString()
```

nodeState nodeState.putShared(String PropertyName, String ProperyValue)

Sets the value of the named shared state property. Note that property names are case-sensitive.

Example:

```
try {
  var currentAuthLevel = nodeState.get("authLevel").asString()
} catch (e) {
  nodeState.putShared("errorMessage", e.toString())
}
```

nodeState nodeState.putTransient(String PropertyName, String ProperyValue)

Sets the value of the named transient state property. Note that property names are case-sensitive.

Example:

```
nodeState.putTransient("sensitiveKey", "sensitiveValue")
```

Accessing profile data

Scripted decision nodes can access profile data through the methods of the `idRepository` object.

Methods

Set idRepository.getAttribute(String UserName, String AttributeName)

Return the values of the named attribute for the named user.

Void idRepository.setAttribute(String UserName, String AttributeName, Array Attribute Values)

Set the named attribute as specified by the attribute value for the named user, and persist the result in the user's profile.

Example:

JavaScript | Groovy

```
var username = nodeState.get("username")
var attribute = "mail"

idRepository.setAttribute(username, attribute,
["user.0@a.com", "user.0@b.com"])
```

Void idRepository.addAttribute(String UserName, String AttributeName, String Attribute Value)

Add an attribute value to the list of attribute values associated with the attribute name for a particular user.

Example:

```
var username = nodeState.get("username")
var attribute = "mail"

// Add a value as a String.
idRepository.addAttribute(username, attribute, "user.0@c.com")
logger.error(idRepository.getAttribute(username,
attribute).toString())
// > ERROR: [user.0@a.com, user.0@c.com]

// Get the first value.
logger.error(idRepository.getAttribute(username,
attribute).iterator().next())
// > ERROR: user.0@a.com

// Get a value at the specified index.
logger.error(idRepository.getAttribute(username,
attribute).toArray()[1])
// > ERROR: user.0@c.com

logger.error(idRepository.getAttribute(username, "non-existing-
attribute").toString())
// > ERROR: []: If no attribute by this name is found, an empty
Set is returned.
```

Setting session properties

Scripted Decision Node scripts can create session properties by using the `Action` interface. The following examples set the outcome to `true`, and add a custom session property:

JavaScript | Groovy

```
var goTo = org.forgerock.openam.auth.node.api.Action.goTo

action =
goTo("true").putSessionProperty("mySessionProperty","myProper
tyValue").build()
```

NOTE

Add the property name to the **Allowlisted Session Property Names** list in the **Session Property Whitelist Service**; otherwise, it will not be added to sessions.

For more information on this service, refer to Session Property Whitelist Service.

Add the script to a scripted decision node in your authentication tree. Users that authenticate successfully using that tree will have the property added to their session, as shown in the following output when introspecting a session:

```
{
    "username": "15249a65-8f9a-4063-9586-a2465963cee4",
    "universalId": "id=15249a65-8f9a-4063-9586-
a2465963cee4,ou=user,o=alpha,ou=services,ou=am-config",
    "realm": "/alpha",
    "latestAccessTime": "2020-10-22T15:01:14Z",
    "maxIdleExpirationTime": "2020-10-22T15:31:14Z",
    "maxSessionExpirationTime": "2020-10-22T17:01:13Z",
    "properties": {
        "AMCtxId": "dffed74d-f203-469c-9ed2-34738915baea-5255",
        "mySessionProperty": "myPropertyValue"
    }
}
```

Accessing existing session properties

Scripted Decision Node scripts can access any existing session properties during a session upgrade request, by using the `existingSession` object.

The following table lists the methods of the `existingSession` object:

Methods

`String existingSession.get(String _Property Name)`

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.

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.

Example:

```
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.

Example:

The following scripts use the `NameCallBack` callback to request a "Nickname" value from the user, and adds the returned value to the `nodeState` object for use elsewhere in the authentication tree:

Groovy | JavaScript

```
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 {
  nodeState.putShared("Nickname", callbacks.get(0).getName())
  action = Action.goTo("true").build()
}
```

For a list of supported callbacks, refer to Supported callbacks.

Accessing credentials and secrets

Scripts used in a scripted decision node can access the secrets configured in AM secret stores *on the file system*.

For example, a script can access credentials or secrets defined in a file system secret volume to make outbound calls to a third-party REST service, without hardcoding those credentials in the script.

Methods

`String secrets.getGenericSecret(String Secret ID)`

Returns the value of the specified secret ID.

If the secret ID is defined at the realm level, its value is returned; otherwise, the script returns the value defined at the global level.

Only secret IDs that begin with the string `scripted.node.` are accessible to scripts.

For more information on creating secret IDs in a secret store, refer to Secret stores.

Use the following functions to format the returned secret value:

`getAsBytes()`

Retrieve the secret value in `byte[]` format.

`getAsUtf8()`

Retrieve the secret value in UTF-8 format.

Example:

The following example scripts show how to get the value (`passwd`) from a secret ID named `scripted.node.secret.id` . They use the value in a basic authentication header to access the http://httpbin.org/basic-auth/{user}/{passwd} service:

JavaScript | Groovy

```
var username = "demoUser"
var password =
secrets.getGenericSecret("scripted.node.secret.id").getA
sUtf8()

var auth =
java.util.Base64.getEncoder().encodeToString(java.lang.S
tring(username + ":" + password).getBytes())
```

```
var request = new org.forgerock.http.protocol.Request()
request.setMethod("GET")
request.setUri("http://httpbin.org/basic-
auth/demoUser/passwd")
request.getHeaders().add("content-
type","application/json; charset=utf-8")
request.getHeaders().add("Authorization", "Basic " +
auth)

var response = httpClient.send(request).get()
var jsonResult =
JSON.parse(response.getEntity().getString())
logger.error("Script result: " +
JSON.stringify(jsonResult))
if (jsonResult.hasOwnProperty("authenticated")) {
    logger.error("outcome = success")
    outcome = "success"
}  else {
    logger.error("outcome = failure")
    outcome = "failure"
}
```

NOTE

To use these sample scripts, you may need to add the following classes to the class allowlist property in the `AUTHENTICATION_TREE_DECISION_NODE` scripting engine configuration:

- `org.mozilla.javascript.ConsString`
- `java.util.Base64`
- `java.util.Base64$Encoder`

Refer to Security.

Adding audit information

Use a script to add information to audit log entries with the `auditEntryDetail` variable.

AM appends the value of the variable to the authentication audit logs.

For JavaScript, the variable must be a string. This example adds the username and email address to the log entry:

```
var username = nodeState.get("username").asString();
var email =
idRepository.getAttribute(username,"mail").iterator().next().toStr
ing();

var detailStr = ("Extra Audit: [" + username + "] Email address: "
+ email).toString();
auditEntryDetail= detailStr;

outcome = "true";
```

AM records the audit string in the `entries > info > nodeExtraLogging > auditInfo` field of the authentication log entry:

```
{
    "_id": "31a9caa2-1439-4088-813f-a60c5f083a45-7725962",
    "timestamp": "2022-12-13T17:22:14.818Z",
    "eventName": "AM-NODE-LOGIN-COMPLETED",
    "transactionId": "31a9caa2-1439-4088-813f-a60c5f083a45-
7725948",
    "trackingIds": [
        "31a9caa2-1439-4088-813f-a60c5f083a45-7725576"
    ],
    "principal": [
        "bjensen"
    ],
    "entries": [
        {
            "info": {
                "nodeOutcome": "true",
                "treeName": "Example",
                "displayName": "Audit Entry",
                "nodeType": "ScriptedDecisionNode",
                "nodeId": "275a8aeb-7a61-4c70-acc2-decb1e0bc139",
                "authLevel": "0",
                "nodeExtraLogging": {
                    "auditInfo": "Extra Audit: [bjensen] Email
address: bjensen@example.com"
                }
            }
        }
    ],
    "realm": "/alpha",
```

```
    "component": "Authentication"
}
```

For more information about auditing, refer to Audit logging.

Authentication module properties

This page provides a reference to configuration properties for AM authentication modules.

Active Directory module properties

`amster` service name: `ActiveDirectoryModule`

`ssoadm` service name: `sunAMAuthADService`

Primary ActiveDirectory Server, Secondary ActiveDirectory Server

Specify the primary and secondary directory server(s).

Both properties take more than one value, allowing more than one primary or secondary remote server, respectively.

Directory servers generally use built-in data replication for high availability. Thus, a directory service typically consists of a pool of replicas to which AM can connect to retrieve and update directory data. AM attempts to contact the primary server(s) first, but if unavailable, AM attempts to contact the secondary servers.

For the current AM server, specify each directory server in the format *`server:port`*. For other AM servers in the deployment, define each server as *`local_server_name`*`|`*`server:port`*. For example, if the *`server`* is `https://openam.example.com:8443/openam`, and the directory server is accessible at `opendj.example.com:1636`, enter the value as `openam.example.com|opendj.example.com:1636`.

Assuming a multi-data center environment, AM determines priority within the primary and secondary remote servers as follows:

- LDAP servers that are mapped to the current AM instance have the 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`.

- LDAP servers that are not specifically mapped to a given AM instance have 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`.

  `ssoadm` attributes are: primary is `iplanet-am-auth-ldap-server`; secondary is `iplanet-am-auth-ldap-server2`.

DN to Start User Search

Specifies the base DN from which AM searches for users to authenticate.

LDAP data is organized hierarchically, similar to 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 start searches from a specific organizational unit, such as `OU=sales,DC=example,DC=com`.

If multiple entries exist with identical search attribute values, ensure this value is specific enough to return a single entry.

`amster` attribute: `userSearchStartDN`

`ssoadm` attribute: `iplanet-am-auth-ldap-base-dn`

Bind User DN, Bind User Password

Specify the user and password of the administration account used for authentication to the directory server.

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.

Make sure that the password is correct before you logout. If it is incorrect, you will be locked out and you will need to log in with the superuser DN. By default, this is `uid=amAdmin,ou=People,`*`AM-deploy-base`*, where *`AM-deploy-base`* was set during AM configuration.

`amster` attributes: `userBindDN` and `userBindPassword`

`ssoadm` attributes: `iplanet-am-auth-ldap-bind-dn` and `iplanet-am-auth-ldap-bind-passwd`

Attribute Used to Retrieve User Profile

LDAP uses this attribute to search for the profile of an authenticated user. Usually, this is the same attribute used to find the user account, such as the value set as the

`uid` in AM. For example, where the attribute is set to `mail`, the LDAP module searches `CN=Users,DC=example,DC=com` with a filter `"(MAIL=bjensen@example.com)"`, and the directory returns the user profile that matches `MAIL=bjensen@example.com`.

The attribute is only used if User Profile is set to `Required` and `Return User DN to DataStore` is not enabled.

`amster` attribute: `userProfileRetrievalAttribute`

`ssoadm` attribute: `iplanet-am-auth-ldap-user-naming-attribute`

Attributes Used to Search for a User to be Authenticated

The attributes specified in this list define the LDAP search filter. 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`.

`amster` attribute: `userSearchAttributes`

`ssoadm` attribute: `iplanet-am-auth-ldap-user-search-attributes`

User Search Filter

The User Search Filter text box provides a more complex filter. For example, if you search using `mail` and add the 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.

`amster` attribute: `userSearchAttributes`

`ssoadm` attribute: `iplanet-am-auth-ldap-search-filter`

Search Scope

This attribute defines the level of directory that will be searched for a matching profile. You can set the search to run at a high level or against a specific area:

- OBJECT searches only for the entry specified as the 'DN to Start User Search'.
- ONELEVEL searches only the entries that are direct children of that object.
- SUBTREE searches the entry specified and all entries at levels below.

`ssoadm` attribute: `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 the server's 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.

To let users change passwords through AM, Active Directory requires that you connect over SSL. The default LDAP port is 389. If you are connecting to Active Directory over SSL, the default LDAPS port is 636.

For SSL or TLS security, enable the SSL/TLS Access to Active Directory Server property.

`ssoadm` attribute: `openam-auth-ldap-connection-mode`

Possible values: `LDAP`, `LDAPS`, and `StartTLS`

Return User DN to DataStore

If User Profile is set to `Required`, this attribute determines whether the DN or the username is returned as the authentication principal.

When enabled, the module returns the DN rather than the User ID or the value set in `Attribute Used to Retrieve User Profile`. The returned value is then used to make the request to retrieve the profile attributes from the user store.

`amster` attribute: `returnUserDN`

`ssoadm` attribute: `iplanet-am-auth-ldap-return-user-dn`

User Creation Attributes

This list of attributes defines the mapping of internal attribute names to external attribute names for dynamic profile creation. The attributes retrieved from the user's authenticated profile are mapped against the values that will be provisioned into their matching account in the data store. This list does not include `uid` mappings.

The format of the list is *`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. 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 seconds, 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`

Stop LDAP Binds after in-memory lockout

If enabled, prevent AM from sending further bind requests to the LDAP Server when the user is locked out through a duration lockout.

amster attribute: `stopLdapbindAfterInmemoryLockedEnabled`

ssoadm attribute: `openam-auth-stop-ldap-bind-after-inmemory-locked-enabled`

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` , 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`

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/security/keys/amster/` directory. 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`

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 a user accesses the default module instance login URL, 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, and its user status must be `Active`.

`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|ldap-server: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 certificate, which can be in one of the following formats:

- Raw PEM-encoded.
- PEM-encoded first, and then URL-encoded.

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, go 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`

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

Specify the primary and secondary directory server(s).

Both properties take more than one value, allowing more than one primary or secondary remote server, respectively.

Directory servers generally use built-in data replication for high availability. Thus, a directory service typically consists of a pool of replicas to which AM can connect to retrieve and update directory data. AM attempts to contact the primary server(s) first, but if unavailable, AM attempts to contact the secondary servers.

For the current AM server, specify each directory server in the format *`server:port`*. For other AM servers in the deployment, define each server as *`local_server_name`*`|`*`server:port`*. For example, if the *`server`* is `https://openam.example.com:8443/openam`, and the directory server is accessible at `opendj.example.com:1636`, enter the value as `openam.example.com|opendj.example.com:1636`.

Assuming a multi-data center environment, AM determines priority within the primary and secondary remote servers as follows:

- LDAP servers that are mapped to the current AM instance have the 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`.

- LDAP servers that are not specifically mapped to a given AM instance have 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`.

  `ssoadm` attributes are: primary is `iplanet-am-auth-ldap-server`; secondary is `iplanet-am-auth-ldap-server2`.

DN to Start User Search

Specifies the base DN from which AM searches for users to authenticate.

LDAP data is organized hierarchically, similar to 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 start searches from a specific organizational unit, such as `OU=sales,DC=example,DC=com`.

If multiple entries exist with identical search attribute values, ensure this value is specific enough to return a single entry.

amster attribute: `userSearchStartDN`

ssoadm attribute: `iplanet-am-auth-ldap-base-dn`

Bind User DN, Bind User Password

Specify the user and password of the administration account used for authentication to the directory server.

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.

Make sure that the password is correct before you logout. If it is incorrect, you will be locked out and you will need to log in with the superuser DN. By default, this is `uid=amAdmin,ou=People,`*`AM-deploy-base`*, where *`AM-deploy-base`* was set during AM configuration.

amster attributes: `userBindDN` and `userBindPassword`

ssoadm attributes: `iplanet-am-auth-ldap-bind-dn` and `iplanet-am-auth-ldap-bind-passwd`

Attribute Used to Retrieve User Profile

LDAP uses this attribute to search for the profile of an authenticated user. Usually, this is the same attribute used to find the user account, such as the value set as the `uid` in AM. For example, where the attribute is set to `mail`, the LDAP module searches `CN=Users,DC=example,DC=com` with a filter `"(MAIL=bjensen@example.com)"`, and the directory returns the user profile that matches `MAIL=bjensen@example.com`.

The attribute is only used if User Profile is set to `Required` and `Return User DN to DataStore` is not enabled.

amster attribute: `userProfileRetrievalAttribute`

ssoadm attribute: `iplanet-am-auth-ldap-user-naming-attribute`

Attributes Used to Search for a User to be Authenticated

The attributes specified in this list define the LDAP search filter. 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`.

amster attribute: `userSearchAttributes`

ssoadm attribute: `iplanet-am-auth-ldap-user-search-attributes`

User Search Filter

The User Search Filter text box provides a more complex filter. For example, if you search using `mail` and add the 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.

`amster` attribute: `userSearchAttributes`

`ssoadm` attribute: `iplanet-am-auth-ldap-search-filter`

Search Scope

This attribute defines the level of directory that will be searched for a matching profile. You can set the search to run at a high level or against a specific area:

- OBJECT searches only for the entry specified as the 'DN to Start User Search'.
- ONELEVEL searches only the entries that are direct children of that object.
- SUBTREE searches the entry specified and all entries at levels below.

`ssoadm` attribute: `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 the server's 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

If User Profile is set to `Required`, this attribute determines whether the DN or the username is returned as the authentication principal.

When enabled, the module returns the DN rather than the User ID or the value set in `Attribute Used to Retrieve User Profile`. The returned value is then used to make the request to retrieve the profile attributes from the user store.

`amster` attribute: `returnUserDN`

`ssoadm` attribute: `iplanet-am-auth-ldap-return-user-dn`

User Creation Attributes

This list of attributes defines the mapping of internal attribute names to external attribute names for dynamic profile creation. The attributes retrieved from the user's authenticated profile are mapped against the values that will be provisioned into their matching account in the data store. This list does not include `uid` mappings.

The format of the list is *internal_attr1*|*external_attr1*.

amster attribute: profileAttributeMappings

ssoadm attribute: iplanet-am-ldap-user-creation-attr-list

Minimum Password Length

Set the minimum length required for a valid password when a user needs to reset their password during authentication.

This value is distinct from password requirements set by the underlying directory server. To avoid confusion, set the value to 0 if an external LDAP server is enforcing password policy.

amster attribute: minimumPasswordLength

ssoadm attribute: iplanet-am-auth-ldap-min-password-length

LDAP Behera Password Policy Support

Determines whether LDAP Behera password policies are supported by a directory server such as ForgeRock Directory Services. If the property is set to false, then only the older Netscape VCHU password policy standard will be enforced.

amster attribute: beheraPasswordPolicySupportEnabled

ssoadm attribute: iplanet-am-auth-ldap-behera-password-policy-enabled

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. 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 seconds, 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`

Stop LDAP Binds after in-memory lockout

If enabled, prevent AM from sending further bind requests to the LDAP Server when the user is locked out through a duration lockout.

`amster` attribute: `stopLdapbindAfterInmemoryLockedEnabled`

`ssoadm` attribute: `openam-auth-stop-ldap-bind-after-inmemory-locked-enabled`

Legacy OAuth 2.0/OpenID Connect authentication module properties

IMPORTANT

This authentication module is labeled as legacy. 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.

For Google implementations, use `_org.forgerock.openam.authentication.modules.oidc.JwtAttributeMapper|*|Google-+`.

For Facebook implementations, use `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper|*|facebook-`.

Default: `org.forgerock.openam.authentication.modules.common.mapping.JsonAttributeMapper`

`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 user status of this anonymous user must be `Active`. 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 OAuth 2.0. 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

AM Field	Details
Authorization Endpoint URL	`/oauth2/authorize` under the deployment URL. 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 Authenticate with a browser, where `module` specifies the authentication module instance to use or `service` specifies the authentication chain to use when authenticating the resource owner. Example: `https://openam.example.com:8443/openam/oauth2/realms/root/authorize`.

AM Field	Details
Access Token Endpoint URL	`/oauth2/access_token` under the deployment URL. 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 Authenticate with a browser, where `module` specifies the authentication module instance to use or `service` specifies the authentication chain to use when authenticating the resource owner. Example: `https://openam.example.com:8443/openam/oauth2/realms/root/access_token`.

AM Field	Details
User Profile Service URL	`/oauth2/tokeninfo` under the deployment URL. Example: `https://openam.example.com:8443/openam/oauth2/realms/root/tokeninfo`.

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 Mitigation 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`

In the AM admin UI, 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

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 of the form `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 of the service account AM uses to authenticate to the directory server. The default is `uid=admin`.

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 Reset registered devices over 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; AM ignores keys specified in JWT headers, such as `jku` and `jwe`.

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 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 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`

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 of the form *`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.

NOTE

AM does not use round-robin load balancing to set priority. AM uses an active-passive algorithm, determining the highest priority to the first available server within the primary server list. If no primary servers are available, AM uses the secondary remote server.

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 admin UI 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 admin UI, go 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 SLO in Integrated Mode (Chains).

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 admin UI 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, go 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, go 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 admin UI, go 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 denylist is applied AFTER the allowlist to exclude those classes. Access to a class specified in both the allowlist and the denylist 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.

By default, the module uses the following TCP/IP ports: `57943`, `58943`.

`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 `/path/to/openam/config/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, go 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

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 user status of this anonymous user must be `Active` . 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, go 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 Mitigation 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

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 user status of this anonymous user must be `Active`. The `Map to anonymous user` property maps authorized users without a profile to this anonymous 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, go 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 Mitigation 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

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

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 user status of this anonymous user must be `Active`. 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

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 user status of this anonymous user must be `Active`. 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

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 user status of this anonymous user must be `Active`. 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

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

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 user status of this anonymous user must be `Active`. 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`.

Learn more in How do I set up the WDSSO authentication module in PingAM 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`

Authenticating to Windows Desktop SSO Using REST

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. For example, if the credentials `demo:Ch4ng31t` are base64-encoded, the resulting string is `ZGVtbzpDaDRuZzMxdA==` .

```
$ 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: Ch4ng31t" \
--header "Authorization: Basic ZGVtbzpDaDRuZzMxdA==" \
'https://openam.example.com:8443/openam/json/realms/root/realms/alpha/authenticate'
{
    "tokenId":"AQIC5w…NTcy*",
    "successUrl":"/openam/console",
    "realm":"/"
}
```

Authentication modules configuration reference

The AM admin UI 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.

Account Active Check module

The 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 fails early without processing modules that appear after the Account Active Check module.

For more information, refer to Account lockout.

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, refer to 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:

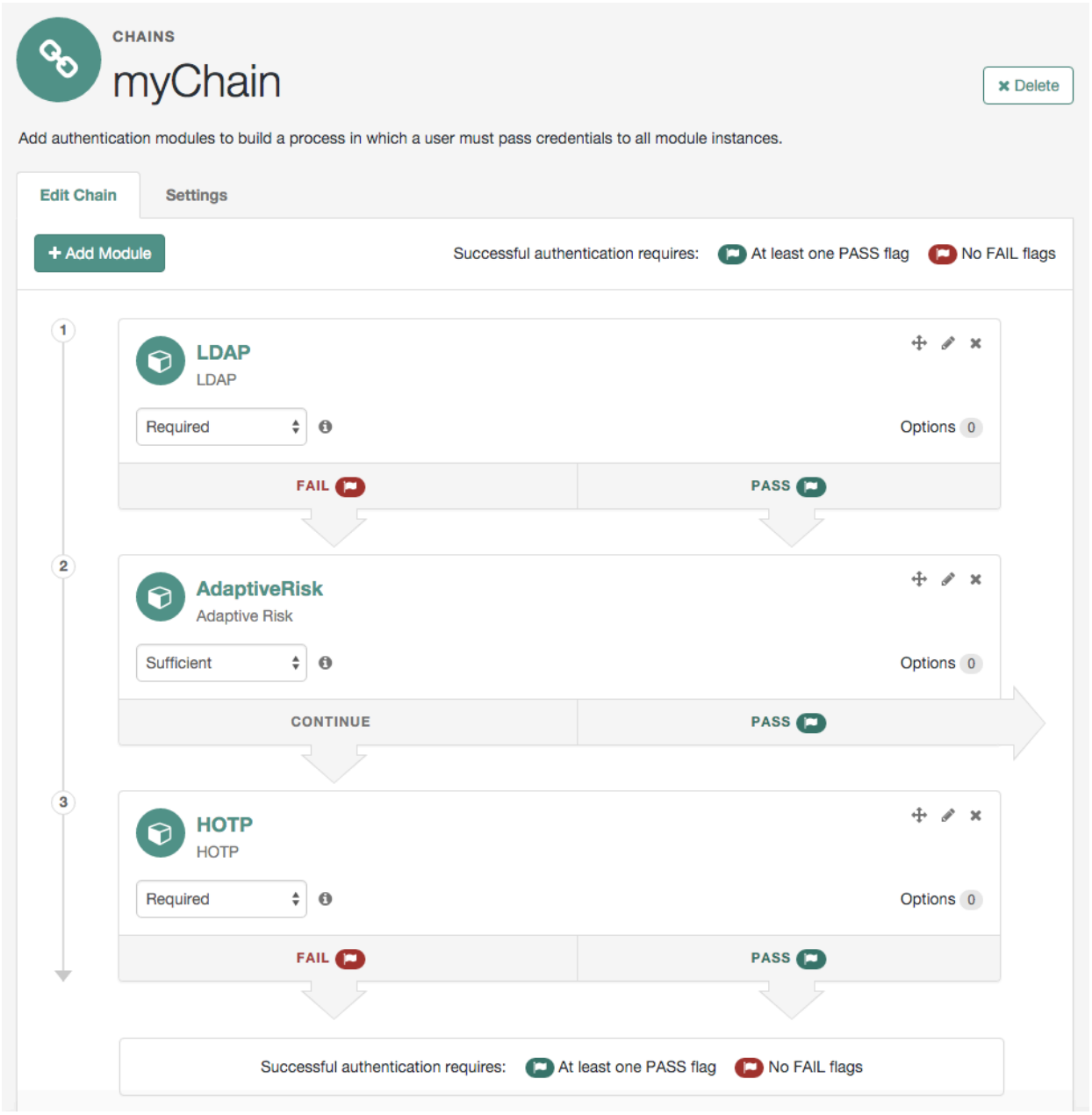


Figure 73. Adaptive Risk Module in an Authentication Chain

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, refer to Handling HTTP Request Headers.

For detailed information about this module's configuration properties, refer to Adaptive Risk authentication module properties.

Amster authentication module

This module lets Amster clients authenticate using established SSH keys.

The Amster client signs a JWT (containing subject and expiration claims) using a local private key. The subject claim is interpreted as the username of the principal. AM verifies the signature, using the list of public keys in its authorized keys file, and finding a key that matches the JWT's key identifier claim. If the entry in the authorized keys file contains a `from` parameter, only connections that originate from the qualifying host are permitted.

For detailed information about this module's configuration properties, refer to Amster 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 these 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.

For detailed information about this module's configuration properties, refer to Anonymous authentication module properties.

Certificate authentication module

X.509 digital certificates can enable secure authentication without the need for usernames 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 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.

You can find more information on configuring certificate authentication in How do I configure certificate-based authentication in PingAM? and How do I troubleshoot certificate-based authentication issues in PingAM?

For detailed information about this module's configuration properties, refer to 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 flat file 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, refer to 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, refer to 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:

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.

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.

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, refer to HOTP authentication module.

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, refer to 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.

Configure the Device ID (Match) authentication module

1. In the AM admin UI, go to **Realms** > ***Realm Name*** > **Authentication** > **Modules**.
   - Click **Add Module**.
   - In the **Module Name** box, enter `Device-ID-Match`.
   - In the **Type** box, select `Device Id (Match)`, and click **Create**.
   - Save your changes.

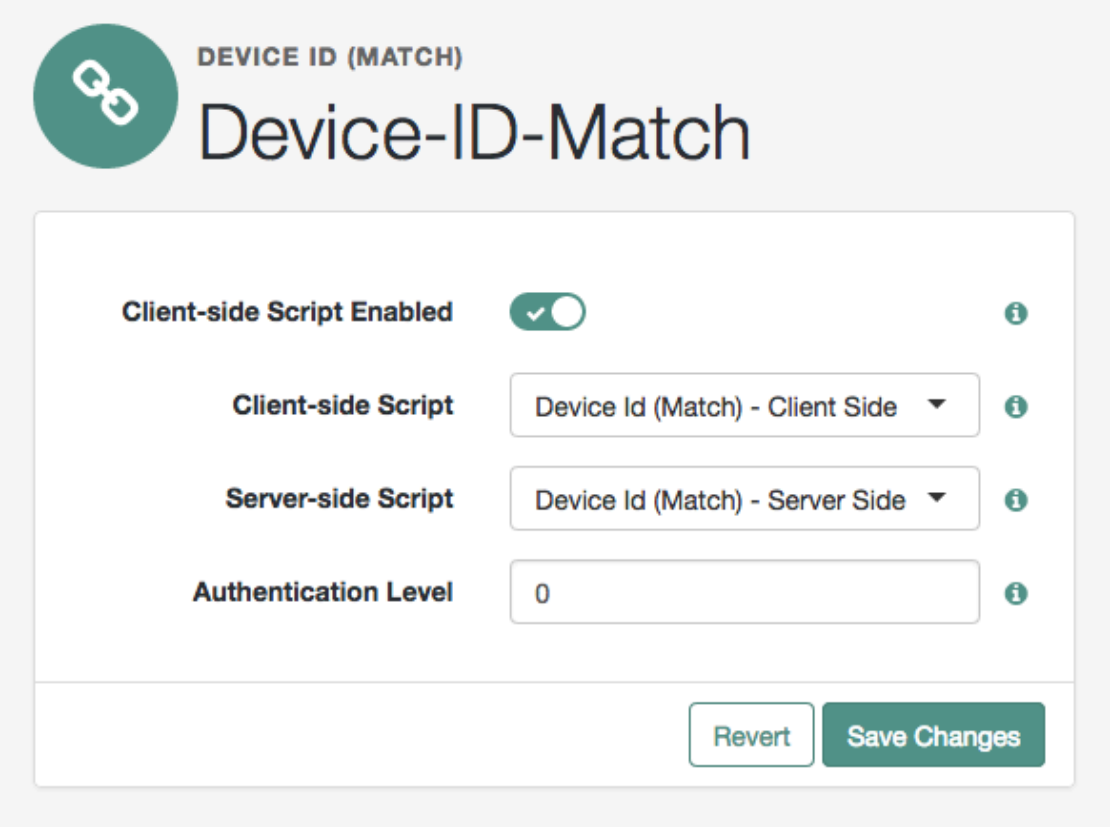


Figure 74. Device ID (Match) Module

2. To make adjustments to the default scripts, click the **Scripts** drop-down list, and click **Device Id (Match) - Client Side**.
3. To make corresponding changes to the server-side script, click the **Scripts** drop-down list, and click `Device Id (Match) - Server Side`.

   For more information, refer to Manage scripts (UI).

Configure an authentication chain with a Device ID (Match) authentication module

1. In the AM admin UI, go to **Realms** > ***Realm Name*** > **Authentication** > **Chains**.
2. On the **Authentication Chains** page, do the following steps:
    - Click **Add Chain**. In the Chain Name box, enter a descriptive label for your authentication chain, and click **Create**.
    - Click **Add Module**.
    - On the **New Module** dialog, select the authentication module, select the criteria, and 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, refer to Configure the Device ID (Match) authentication module and To Configure the Device ID (Save) authentication module.
3. Review your authentication chain, and save your changes.

What the user sees during authentication

When the user logs on to the AM admin UI, 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 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, refer to 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. In the AM admin UI, go to **Realms** > ***Realm Name***, > **Authentication** > **Modules**.
2. To add the Device ID (Save) module, click **Add Module**.
3. In the **Module Name** box, enter `Device-ID-Save` .
4. In the **Type** box, select `Device Id (Save)` , and click **Create**.
5. To configure the Device-Id (Save) module, do the following:
    - 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.
    - In the **Maximum stored profile quantity** box, enter the max number of stored profiles. Any profile that exceeds this number will not be stored.
    - In the Authentication Level box, enter a number corresponding to the authentication level of the module.
    - Click **Save Changes**.

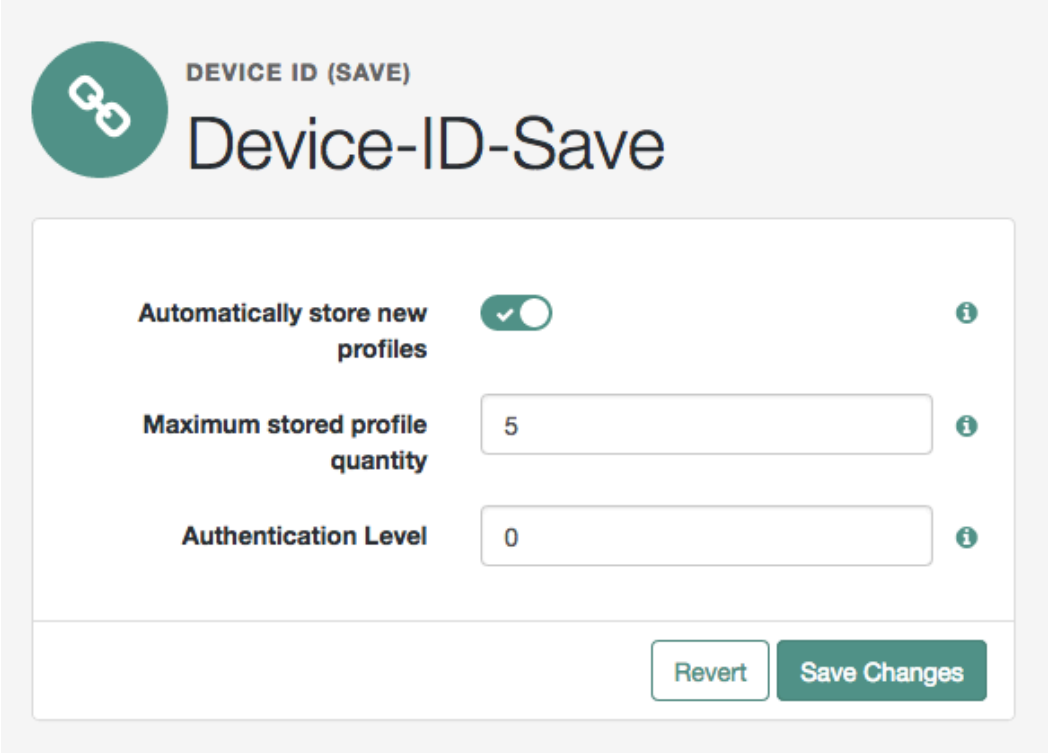


Figure 75. Device ID (Save) Module

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, refer to Federation authentication module properties.

NOTE

When configuring AM to use WS-Federation, add the hosted and remote entity SP/RP token issuer endpoint URLs to the hosted IDP's `Valid WReply List` parameter to ensure successful validation of the `wreply` URL.

You can find information on implementing authentication using WS-Federation in How do I configure PingAM as an Identity Provider for Microsoft Office 365 and Azure using WS-Federation? in the *Knowledge Base*.

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 this module's configuration properties, refer to ForgeRock Authenticator (OATH) authentication module properties.

NOTE

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 Comparing the ForgeRock Authenticator (OATH) to the HOTP authentication module 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, refer to Multi-factor authentication.

For detailed information about this module's configuration properties, refer to 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, refer to Manage devices for MFA.

For detailed information about this module's configuration properties, refer to 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 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, refer to 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 Comparing the ForgeRock Authenticator (OATH) to the HOTP authentication module 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, refer to 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, refer to 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, refer to LDAP authentication module properties.

Legacy OAuth 2.0/OpenID Connect authentication module

NOTE

This authentication module is labeled as *legacy*.

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 an 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, refer to 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, refer to 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 resynchronizes 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 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 log in 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 log in until you reset the counter in AM to match their device. Refer to Reset registered devices over 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 user's 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 HOTP or TOTP.

For detailed information about this module's configuration properties, refer to OATH authentication module properties.

NOTE

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 Comparing the ForgeRock Authenticator (OATH) to the HOTP authentication module 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. For more information, refer to Social authentication.

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:


```
{
    "at_hash":"GZofHqaewBJ2hRjkKiT8Ew",
    "sub":"demo",
    "auditTrackingId":"0d836a8d-af4c-4b33-a3a2-bf06cd590b91-3937",
    "iss":"https://openam.example.com:8443/openam/oauth2",
    "tokenName":"id_token",
    "aud":"myClientID",
    "azp":"myClientID",
    "auth_time":1538675226,
    "name":"demo",
    "realm":"/",
    "exp":1538733152,
    "tokenType":"JWTToken",
    "family_name":"demo",
    "iat":1538675226,
    "jti":"bb18a404-f08c-446e-8e01-8b70c8d48192"
}
```


The `azp`, `aud`, and `iss` values are literally reused in the module configuration. The following figure shows an example configuration for this `id_token` format.

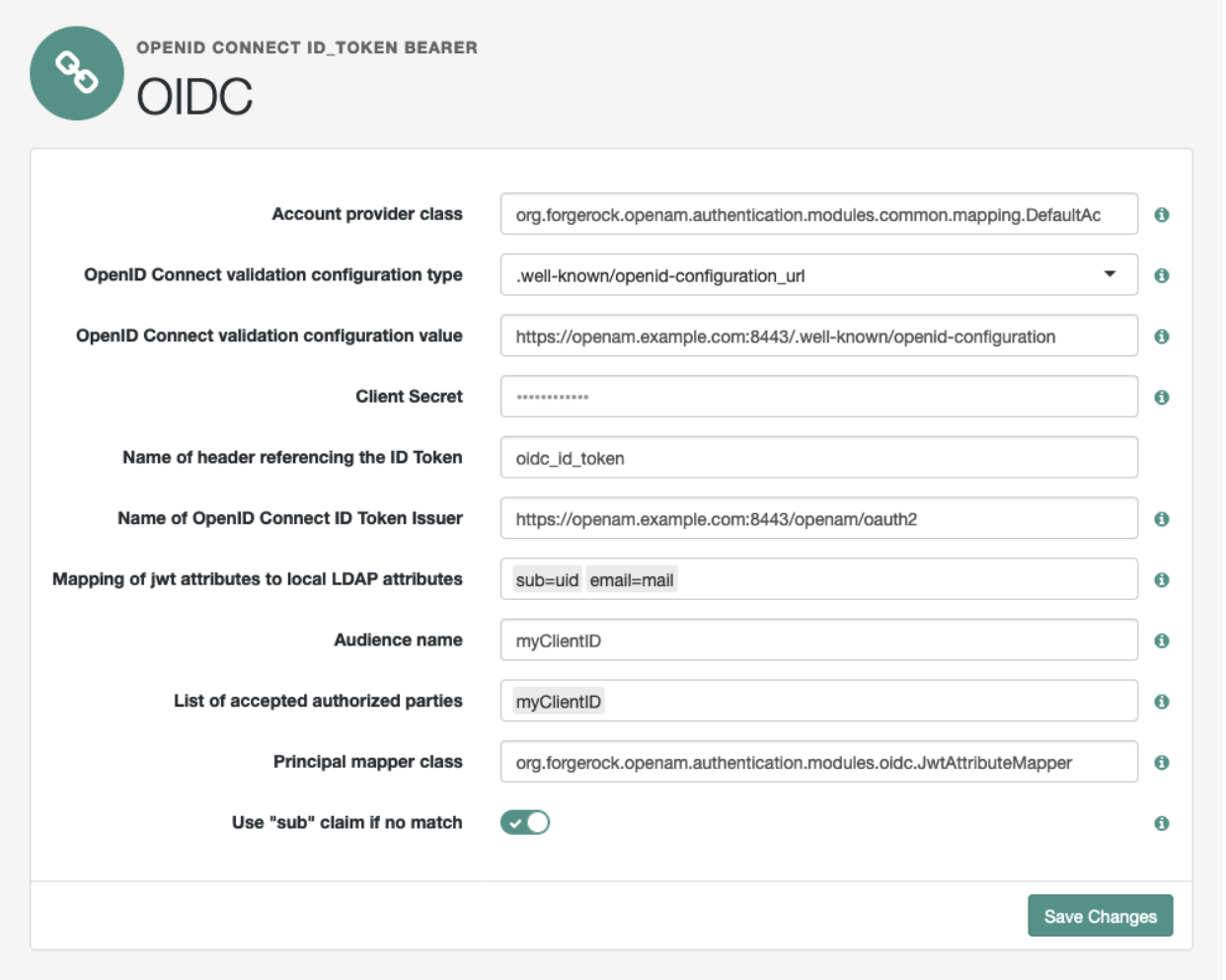


Figure 76. Sample OpenID Connect id_token Bearer Module Configuration

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/realms/alpha/authenticate?authIndexType=module&authIndexValue=OIDC'
{
    "tokenId": "nIq…AA*",
    "successUrl": "/openam/console",
    "realm": "/alpha"
}
```

The `id_token` value, abbreviated as `eyJ…ifQ.eyJ…In0.BT1…iZA`, is the value of the `oidc_id_token` header in the configuration. The request targets a module named `OIDC` as specified by the `authIndexType` and `authIndexValue` parameters. For detailed information about the authentication REST API, refer to Authenticate over REST.

For detailed information about this module's configuration properties, refer to 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. 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 secret ID mappings used to encrypt and then sign persistent cookies:

Secret ID	Default alias	Algorithms
`am.default.authentication.modules.persistentcookie.encryption`	`test`	RSA (at least 2048 bits)
`am.default.authentication.modules.persistentcookie.signing`	`hmacsigningtest`	HS256

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`

Secret ID	Default alias
`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, refer to Mapping and Rotating Secrets.

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, refer to Handling HTTP Request Headers.

The Persistent Cookie module belongs with a second module in an authentication chain. To understand how this works, go 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:

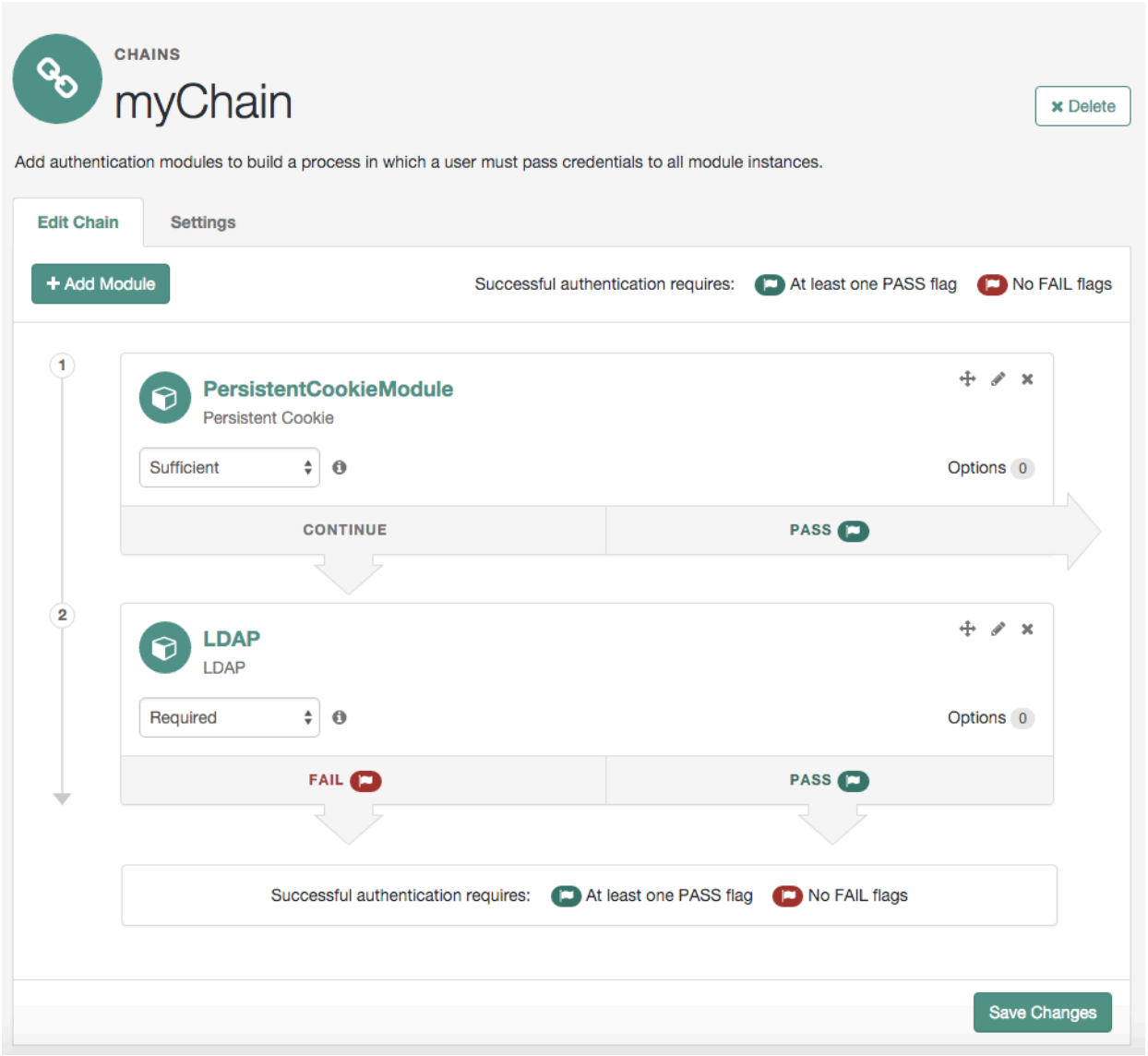
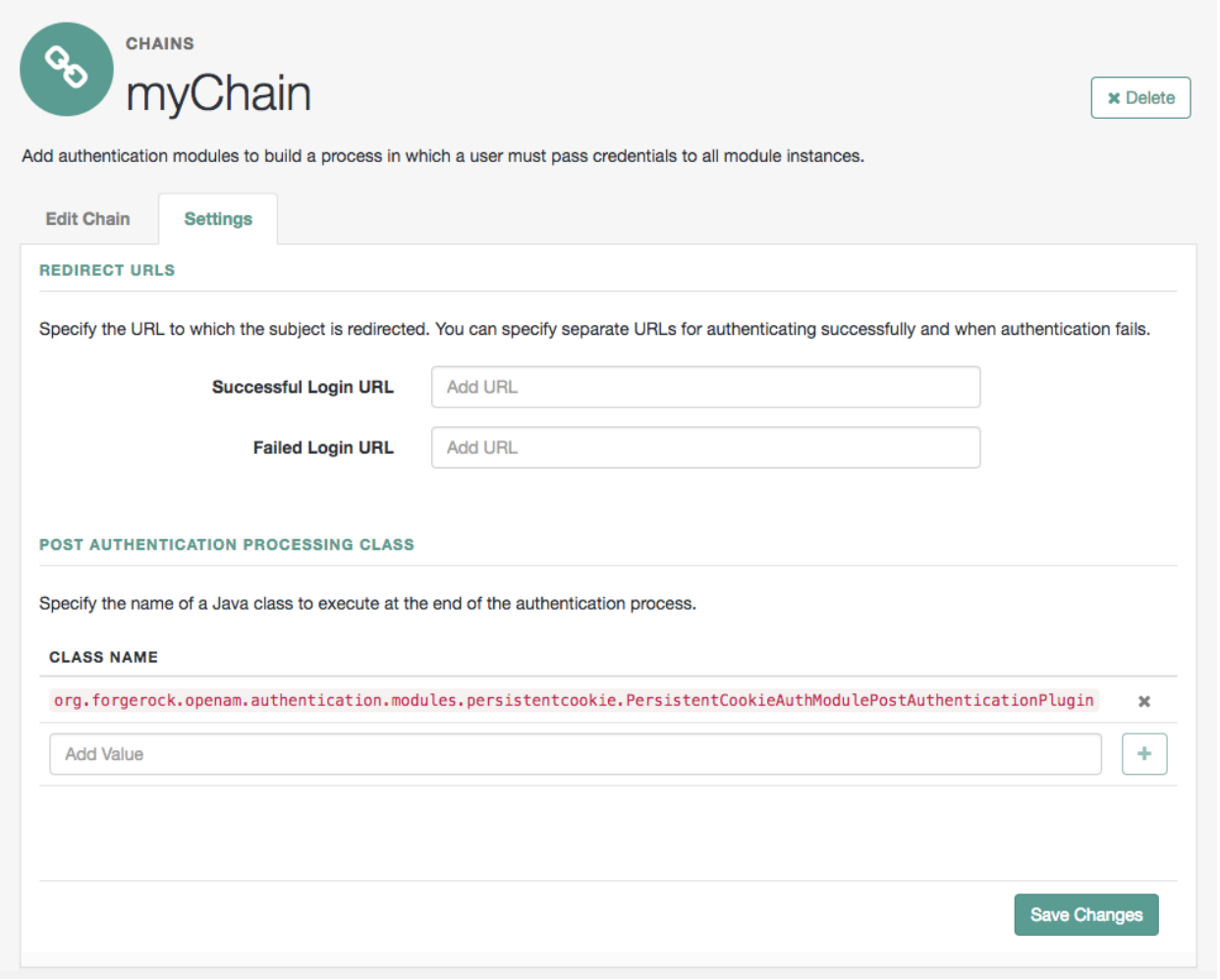


Figure 77. Persistent Cookie Module in an Authentication Chain

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:



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 admin UI, go to **Configure > Authentication**, and click **Persistent Cookie**.

For detailed information about this module's configuration properties, refer to 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, refer to 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, refer to 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 Implement SSO in integrated mode (chains) and make sure you have made all required changes to your service provider configuration.

For detailed information about this module's configuration properties, refer to 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, and 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 admin UI, 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 uid=amAdmin,ou=People,dc=openam,dc=forgerock,dc=org \
--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 uid=amAdmin,ou=People,dc=openam,dc=forgerock,dc=org \
--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 uid=amAdmin,ou=People,dc=openam,dc=forgerock,dc=org \
--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, refer to Using server-side authentication Scripts in authentication modules.

For detailed information about this module's configuration properties, refer to 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.

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.

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, refer to 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 Microsoft Edge to benefit from single sign-on when logged on to a Windows desktop.

For detailed information about this module's configuration properties, refer to 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 websites. For information on a possible workaround, refer to Microsoft knowledge base article KB251404.

Scripted module API

In addition to the functionality provided by Accessing HTTP Services and Debug logging, authentication modules that use server-side scripts can access the authorization state of a request, the information pertaining a session, and the login request itself.

Client-side scripts in modules gather data into a string, which is returned to AM in a self-submitting form.

TIP

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.

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

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.
`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 | Absolute URL

```
...
sharedState.put("gotoOnFailureUrl","/am/XUI/?
service=testChain#failedLogin");
```

```
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.

Was this helpful?