User Guide

/ Java Agents 5.6

Latest update: 5.6.3

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

Abstract

Guide to installing and managing ForgeRock® Access Management Java agents. ForgeRock Access Management provides open source authentication, authorization, entitlement, and federation software.

Table of Contents

Preface

This guide shows you how to install ForgeRock Access Management Java agents, as well as how to integrate with ForgeRock Access Management. Read the Release Notes before you get started.

This guide is written for anyone installing Java agents to interface with supported Java web application containers.

About ForgeRock Identity Platform™ Software

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

Chapter 1

Introducing Java Agents

A *Java agent* is an Access Management add-on component that operates as a Policy Enforcement Point (PEP) or policy agent for applications deployed on a Java container.

Java agents intercept inbound requests to applications. Depending on the *filter mode* configuration, Java agents interact with AM to:

- Ensure that clients provide appropriate authentication.
- Enforce AM resource-based policies.

This chapter covers how Java agents work and how their features can protect your applications.

Java Agent Components

Java agents comprise two main components; the agent filter and the agent application:

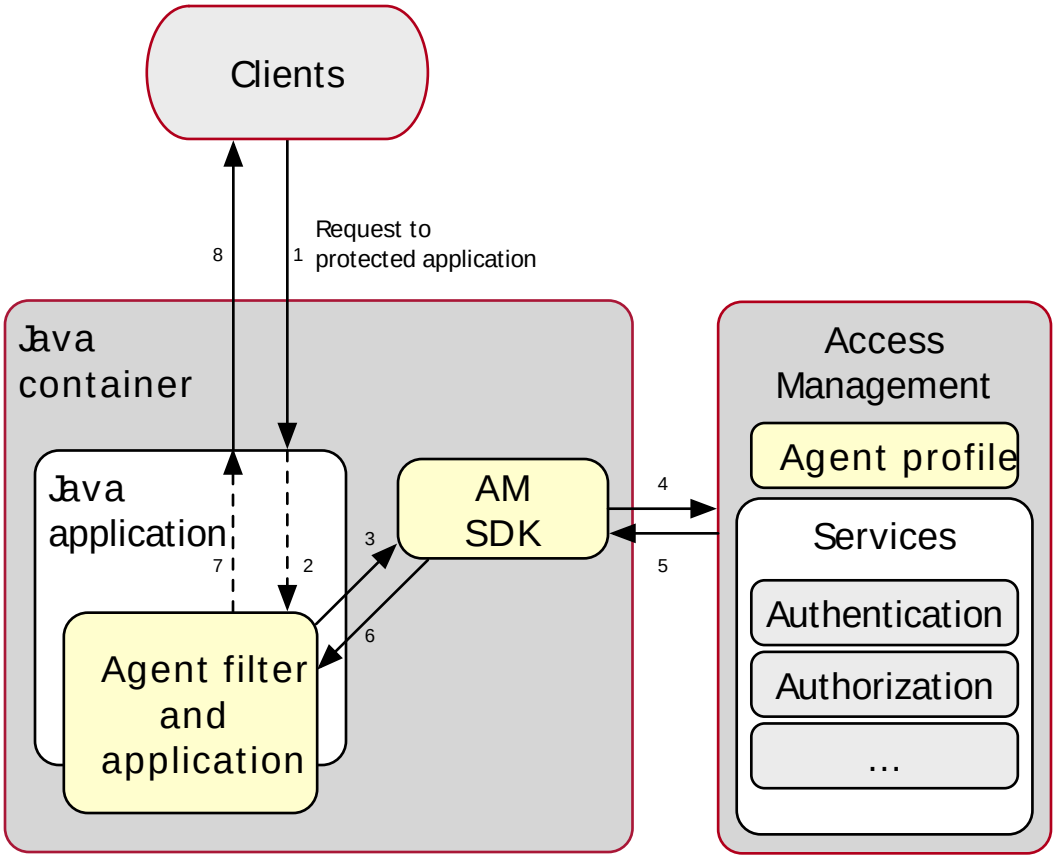
- **Agent Filter**. Intercepts inbound client requests to a resource and processes them based on the filter mode of operation.
- **Agent Application**. Deployed as `agentapp.war`, it is required for authentication and the cross-domain single sign-on (CDSSO) flow.

The following components are not strictly part of the Java agent, but they play an important part in the agent's operation:

- **AM SDKs**. Provide a set of APIs required to interact with AM.
- **Agent Profile**. Contains a set of configuration properties that define the agent's behavior. The agent profile can be stored in AM's configuration store or as a text file local to the agent installation.

The following picture illustrates the Java agent's components when the agent profile is stored in the AM configuration store:

Java Agent Main Components



Configuration Location

Java agents have two sets of properties:

Configuration properties

Configuration properties determine the behavior of the Java agent. AM stores configuration properties either centrally or locally:

- **Centralized configuration**

AM stores the Java agent properties in the AM configuration store. Storing the agent configuration centrally allows you to configure your agents by using the AM console, the **ssoadm** command, and the REST API. This is the default configuration location mode.

To access the centralized agent configuration in the AM console, navigate to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name*.

For more information on creating centrally-stored agent profiles, see "Creating Agent Profiles".

- **Local configuration**

  The Java agent installer creates the `/path/to/java_agents/agent_type/agent_instance/OpenSSOAgentConfiguration.properties` file to store configuration properties locally. To manage the configuration, edit the file to add properties, remove properties, and change values. You cannot update this file using the AM console, the **ssoadm** command, or the REST API.

Bootstrap properties

Bootstrap properties enable Java agents to connect to an AM instance. These properties are required regardless of whether the configuration properties are stored centrally in AM or locally on the agent installation.

The agent installer creates the `/path/to/java_agents/agent_type/agent_instance/config/OpenSSOAgentBootstrap.properties` file, which contains the bootstrap properties.

For more information on setting the `Location of Agent Configuration Repository`, see Profile Properties.

Request Process Flow

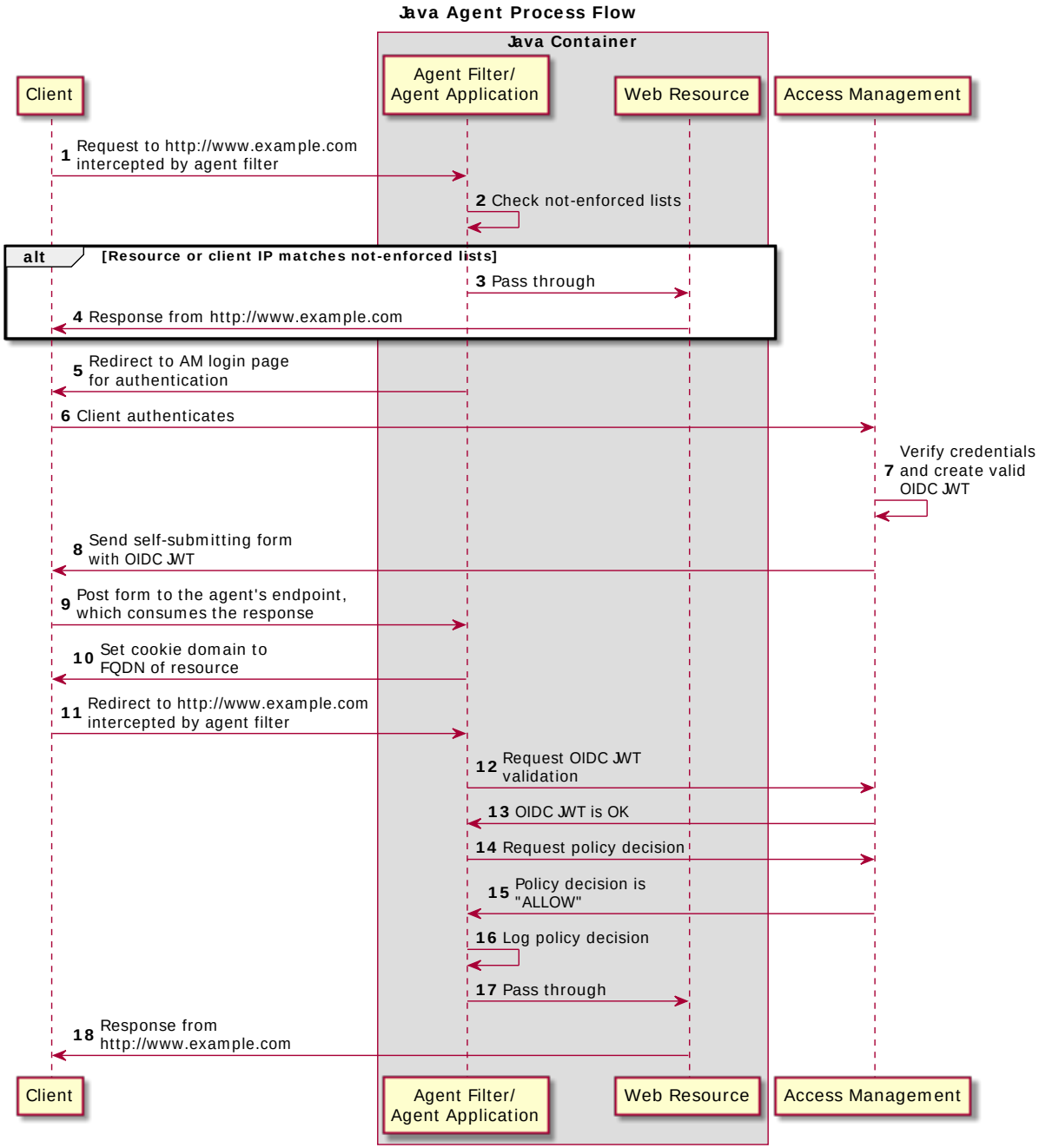
Suppose you wanted to withdraw money from your bank account using an ATM. The ATM would not allow you to access your account unless you identified yourself to the bank with your card and PIN number. For a joint account, you may also require additional authorization to access the funds.

Java agents work on a similar premise. When a client requests access to an application resource, the Java agent intercepts the request. Then, AM validates the identity of the client as well as their authorization to access the protected resource.

The following sequence diagram shows the flow that occurs when an unauthenticated client requests a resource protected by a Java agent and AM. The diagram assumes that the filter mode is set to `ALL` and is simplified [1] to show only the relevant steps in the flow.

[1] For a detailed diagram, see About Cross-Domain Single Sign-On in the *ForgeRock Access Management Authentication and Single Sign-On Guide*.

Java Agent Process Flow



1. An unauthenticated client attempts to access a resource at `www.example.com`. The agent filter intercepts the inbound request.
2. Java agents evaluate whether the requested resource or the client IP address matches any rule contained in the *not-enforced lists*.
3. *Alternate Flow*. The requested resource or the client IP address matches a not-enforced rule. The Java agent allows access to the resource.
4. *Alternate Flow*. The client receives a response from `www.example.com`. The flow ends.
5. The requested resource or the client IP address does not match a not-enforced rule. The Java agent redirects the client to log in to AM.
6. The client authenticates to AM.
7. AM's Authentication Service verifies the client's credentials and creates a valid OpenID Connect (OIDC) JSON Web Token (JWT) with session information.
8. AM sends the client a self-submitting form with the OIDC JWT.
9. The client posts the self-submitting form to the agent's endpoint, and the Java agent consumes it.
10. The Java agent sets the cookie domain to the FQDN of the resource.
11. The client attempts to access the protected resource again, and the Java agent filter intercepts the request.
12. The Java agent contacts AM to validate the session contained in the OIDC JWT.
13. AM validates the session.
14. The Java agent contacts AM's Policy Service, requesting a decision about whether the client is authorized to access the resource.
15. AM's Policy Service returns `ALLOW`.
16. The Java agent writes the policy decision to the audit log.
17. The Java agent enforces the policy decision. Since the Policy Service returned `ALLOW`, the Java agent performs a pass-through operation to return the resource to the client.
18. The client accesses the resource at `www.example.com`.

Java Agent Features

Java agents provide a number of features to help you protect your applications. The following table contains a list of the features and the sections were you can have more information about each one:

- Not-Enforced Lists
- Notification System
- Attribute Fetch Modes
- Login Attempt Limits
- FQDN Checking
- Cookie Reset Properties
- Cross-Domain Single Sign-On
- POST Data Preservation
- Continuous Security
- Redirection and Conditional Redirection
- Caching Capabilities
- Query Parameter Handling

Not-Enforced Lists

Java agents provide the capability to bypass authentication and grant immediate access to resources not requiring protection, thus speeding up agent operation.

You can configure different lists of not-enforced rules depending on the needs of your deployment:

- **Not-Enforced URI Lists**

  Configure not-enforced URI lists to allow access to resources, such as images, stylesheets, or the HTML pages that comprise the public front end of your site.

- **Not-Enforced IP Lists**

  Configure not-enforced IP lists to allow access to your site from an administrative IP address, an internal network range, or a search engine.

- **Compound Not-Enforced URI and IP Lists**

  Configure compound URI and IP not-enforced lists when you require more control over access.

To evaluate access, the Java agent constructs a list of compound rules, a list of simple URI rules, and a list of simple IP rules. The lists are evaluated in the following order:

1. Compound rules in both Not-Enforced URIs and Not-Enforced Client IP List properties

2. IP rules in the Not-Enforced Client IP List property

3. URI rules in the Not-Enforced URIs property

The first time the Java agent receives a request for a resource, it needs to evaluate if the request is for a protected resource or for a not-enforced resource. To make this decision, the agent tries to match the request with the patterns specified in the not-enforced lists.

The Java agent evaluates every rule in the lists in order until it finds the first match. It does not process any other rule, even though a rule further down the list might provide a better match. Because of this, place your most specific rules at or near the beginning of the list.

To speed up future requests, the Java agent caches whether the resource hit or miss any not-enforced rule. Therefore, if a request for the same resource reaches the agent again, the agent checks the result of the rules evaluation in the cache instead of running the rules again.

If no rule matches, the Java agent decides whether to grant access or defer to AM based on the configuration of the Invert Not-Enforced IPs and the Invert Not-Enforced URIs properties. See the following table for an analysis of the possibilities.

Not-Enforced Default Access for Non-Matching Requests

	Not-Enforced Client IP List Property	**Not-Enforced URIs Property**	**Outcome**
Inverted?	No	No	Defer to AM
Inverted?	Yes	Yes	Grant access
Inverted?	Yes	No	Defer to AM
Inverted?	No	Yes	Defer to AM

In the preceding table, if the Not-Enforced Client IP List and Not-Enforced URIs properties are not inverted (the Not-Enforced IP Invert List and Invert Not-Enforced URIs properties are set to `false`), the Java agent defers any unmatched request to AM for authorization.

Not-Enforced lists support wildcards, regular expressions, and the possibility of specify HTTP methods for fine-tuning the rules.

For more information about configuring not-enforced lists and other related properties, see Not-Enforced URI Processing Properties.

Notification System

AM can notify Java agents of configuration and session state changes through WebSockets. Java agents can subscribe to up to three notification feeds:

- **Configuration Notifications**. When the administrator makes a change to a hot-swappable Java agent configuration property, AM sends a notification to the agent to reread the agent profile from AM.

Configuration notifications are applicable when you store the agent profile in AM's configuration data store. For more information about enabling configuration notifications, see Profile Properties.

- **Session Notifications**. When a client logs out or a CTS-based session expires, AM sends a notification to the Java agent to remove that entry from the session cache. For more information about enabling session notifications, see Session Client Service Properties.
- **Policy Notifications**. When an administrator changes a policy, AM sends a notification to the Java agent to flush the policy cache. For more information about enabling policy notifications, see Policy Client Service Properties.

The AM advanced server configuration property, `org.forgerock.openam.notifications.agents.enabled`, controls whether the AM server sends notifications to connected Java agents. This property is enabled by default.

Enabling notifications affects the validity of the Java agent caches. For more information, see "Caching Capabilities".

Note

Ensure that load balancers and reverse proxies configured in your environment support WebSockets.

Attribute Fetch Modes

Java agents provide the capability to fetch and inject user information into HTTP headers, request objects, and cookies and pass them on to the protected client applications. The client applications can then personalize content using these attributes in their web pages or responses.

Specifically, you can configure the type of attributes to be fetched and the associated mappings for the attributes names used on AM to those values used in the containers. The Java agent securely fetches the user and session data from the authenticated user as well as policy response attributes.

For more details, see Session Attributes Processing Properties.

Login Attempt Limits

When the client does not present a valid SSO token, the Java agent will redirect the user to the login URL configured in AM. The Java agent can be configured to limit the login attempts made to the Java agent to mitigate any redirect loops that may result in an error page presented to the end-user.

You can use the `com.sun.identity.agents.config.login.attempt.limit` property to specify a non-zero value for the number of login attempts. For example, if the property is set to 3, then the Java agent will block the access request to the protected resource on the fourth login request.

You can also limit the number of redirections the Java agent can take for a single browser session by setting the `com.sun.identity.agents.config.redirect.attempt.limit`.

For more details, see General Properties.

FQDN Checking

Java agents require that clients accessing protected resources use valid URLs with fully qualified domain names (FQDNs). If invalid URLs are referenced, policy evaluation can fail as the FQDN will not match the requested URL, leading to blocked access to the resource. Misconfigured URLs can also result in incorrect policy evaluation for subsequent access requests.

There are cases where clients may specify resource URLs that differ from the FQDNs stored in AM policies; for example, in load balanced and virtual host environments. To handle these cases, the Java agent supports FQDN Checking properties: `FQDN Default` and `FQDN Virtual Host Map` properties.

The `FQDN Default` property specifies the default URL with valid hostname. The property ensures that the Java agent can redirect to a URL with a valid hostname should it discover an invalid URL in the client request.

The `FQDN Virtual Host Map` property stores map keys and their corresponding values, allowing invalid URLs, load balanced URLs, and virtual host URLs to be correctly mapped to valid URLs. Each entry in the Map has precedence over the `FQDN Default` setting, so that if no valid URLs exist in the `FQDN Virtual Host Map` property, the Java agent redirects to the value specified in the `FQDN Default` property.

If you want the Java agent to redirect to a URL other than the one specified in the `FQDN Default` property, then it is good practice to include any anticipated invalid URLs in the `FQDN Virtual Host Map` property and map it to a valid URL.

For more details, see Fully Qualified Domain Name Checking Properties.

Cookie Reset Properties

AM provides cookie reset properties that the Java agent carries out prior to redirecting the client to a login page for authentication.

Cookie reset is typically used when multiple parallel authentication mechanisms are in play with the Java agent and another authentication system. The Java agent can reset the cookies set by the other mechanism before redirecting the client to a login page.

The cookie reset properties include a name list specifying all of the cookies that will reset, a domain map specifying the domains set for each cookie, and a path map specifying the path from which the cookie will be reset.

If you have enabled attribute fetching using cookies to retrieve user data, it is good practice to use cookie reset, which will reset once you want to access an enforced URL without a valid session.

For more details, see Cookie Reset Properties.

Cross-Domain Single Sign-On

Cross-domain single sign-on (CDSSO) is an AM capability that lets users access multiple independent services from a single login session, using the Java agent to transfer a validated session ID on a single DNS domain or across domains.

Without AM's CDSSO, single sign-on cannot be implemented across domains; the session cookie from one domain would not be accessible from another domain. For example, in a configuration where the AM server (`openam.example.com`) is in a different DNS domain than the Java agent (`myapp.website.com`), single sign-on would not be possible.

Java agents work in CDSSO mode by default, regardless of the DNS domain of the AM servers and the DNS domain of the agents.

For more information and implementation details, see About Single Sign-On and Configuring Cross-Domain Single Sign-On in the *ForgeRock Access Management Authentication and Single Sign-On Guide*.

POST Data Preservation

Java agents can preserve HTML form data submitted as an HTTP POST by unauthenticated clients.

At a high level, when an unauthenticated client posts HTML form data to a protected resource, the Java agent stores the data in its cache and redirects the client to the login screen. Upon successful authentication, the agent recovers the data stores in the cache and autosubmits it to the protected resource.

Consider enabling POST data preservation if users or clients in your environment submit large amounts of data, such as blog posts and wiki pages, and their sessions are short-lived.

Java agents guarantee the integrity of the data and the authenticity of the client as follows:

- Each unauthenticated form POST to a protected resource generates a random unique identifier as the dummy internal endpoint from which the client recovers the POST data after authentication. This identifier is then placed into an encrypted cookie and provided to the client.
- During authentication, the client is provided with a one-time code placed in a different cookie that is also stored with the POST data in the cache. If the client cannot provide the code (because the cookie is missing) or the code differs from the one stored with the POST data, the Java agent denies access to the endpoint.

To mitigate against DoS attacks, manage the time the data lives in the cache and the size of the cache itself, either by limiting the total number of entries it can hold or the total size of the data held.

For more information about the POST data preservation cache and its properties, see "Caching Capabilities" and POST Data Preservation Properties.

Continuous Security

Because Java agents are the first point of contact between users and your business applications, they can collect inbound login requests' cookie and header information which an AM server-side authorization script can then process.

For example, you may decide that only incoming requests containing the `InternalNetwork` cookie can access intranet resources outside working hours.

For more information about configuring continuous security properties, see Continuous Security Properties.

Redirection and Conditional Redirection

Java agents provide the capability to redirect users to a specific AM instance, an AM site, or a website other than AM. You can also redirect users based on the incoming request URL. Conditional redirection is available for login and logout requests.

For example, you can configure the Java agent such that any login request made from the `france.example.com` domain is redirected to the `openam.france.example.com` AM site. You can also configure the Java agent to redirect any user to a specific page after logout.

You may also decide to configure conditional login redirection to specify the realm to which users must authenticate.

Java agents support the following redirection modes:

- Default Redirection Login Mode
- Custom Redirection Login Mode

Default Redirection Login Mode

By default, Java Agents 5.x and AM use OpenID Connect (OIDC) JSON web tokens (JWT) for authentication. Unauthenticated users are redirected to the `oauth2/authorize` endpoint. This endpoint invokes both the XUI and other endpoints within AM, such as:

- `oauth2/authorize`
- `json/authenticate`
- `json/sessions`
- `json/serverinfo`
- `XUI/*`

Unauthenticated users must be able to reach, at least, AM's `oauth2/authorize` endpoint.

When configuring the default redirection login mode, consider the following points:

- **Ensure that the Allow Custom Login Mode property is disabled** (`org.forgerock.openam.agents.config.allow.custom.login` is set to `false`).
- **Configure the following property:**

- AM Conditional Login URL (`org.forgerock.openam.agents.config.conditional.login.url`)

For more information, see Login URL Properties.

- **The login flow is as follows:**

1. The agent receives a request to access a page from an unauthenticated user.
2. The agent matches the request with the domains and URLs specified by the `org.forgerock.openam.agents.config.conditional.login.url` property, and redirects the user to the appropriate custom login page.

   During the redirection process, the agent appends a number of OIDC parameters to the request[2].
3. The user logs in to the custom login page.
4. The custom login page redirects back to the agent and provides, at least, the OIDC parameters appended during the redirection process.
5. The agent contacts AM to log the user into the appropriate realm.

Custom Redirection Login Mode

Java Agents support a custom login redirection mode by configuring the custom login mode property `org.forgerock.openam.agents.config.allow.custom.login`.

When this property is set to `true`, the agent expects the custom login page to set an SSO token in the user's browser after authentication. The agent will present the SSO token to AM, which would then convert it into an OIDC JWT.

Use the custom redirection login mode when:

- Your environment has customized login pages that expect user sessions to be stored in SSO tokens instead of in OIDC JWTs.
- Your environment is configured so the users cannot access the AM servers directly.
- Your environment is configured so the custom login pages are not part of AM's XUI.

Note

You should use the default redirection login mode when designing new environments. The custom redirection login mode is meant as an aid to support environments upgrading from earlier versions of the agents.

When configuring the custom redirection login mode, consider the following points:

[2] For more information, see the implementation details included in the AM Conditional Login URL property.

- **Ensure that the Allow Custom Login Mode property is enabled** (the `org.forgerock.openam.agents.config.allow.custom.login` property is set to `true`).
- **Configure the public AM URL** in the `org.forgerock.agents.public.am.url` bootstrap property if the custom pages are in a network that can only access AM using a proxy, a firewall, or any other technology that remaps the AM URL to one accessible by the custom login pages.

  Consider an example where the traffic between AM and the agent happens through the *example-internal.com* network, but the custom login pages are on the *example-external.com* domain. In this case, you would configure *https://openam.example-external.com:8443/openam* as the public AM URL.
- **Configure one of the following properties:**
  - AM Login URL (`com.sun.identity.agents.config.login.url`)
  - `org.forgerock.openam.agents.config.conditional.custom.login.url`

  For more information, see Login URL Properties.
- **The login flow is as follows:**
  1. The agent receives a request to access a page from an unauthorized user.
  2. The agent checks the custom login redirection mode properties:
     - If configured, the agent redirects the user to the custom login page specified by the AM Login URL property.
     - If not configured, the agent matches the request with the domains and URLs specified by the `org.forgerock.openam.agents.config.conditional.custom.login.url` property, and redirects the user to the appropriate custom login page.

     During the redirection process, the agent appends a `goto` parameter and a nonce to the request.
  3. The user logs in to the custom login page.
  4. The custom login page sets an SSO token in AM's session cookie (by default, `iPlanetDirectoryPro`) in the user's browser and redirects back to the agent using the `goto` parameter provided.

     If the agent is unable to access AM's session cookie, or if the session cookie contains an invalid SSO token, the login process will fail.
  5. The agent contacts AM to log the user in to the appropriate realm and convert the SSO token into an OIDC JWT.

Caching Capabilities

Java agents allocate memory from the Java heap space in the web container to the following caches:

Configuration Cache

When a Java agent with centralized configuration starts up, it makes a call to AM to retrieve a copy of the Java agent profile and stores it in the cache. The information stored in the cache is valid until one of the following events occurs:

- AM notifies the Java agent of changes to hot-swappable Java agent configuration properties. The agent flushes the configuration cache and rereads the agent profile from AM.
- The Java agent restarts.
- The Java agent rereads the configuration from AM or from local files at the frequency specified by the `com.sun.identity.agents.config.load.interval` property.

If notifications and the `com.sun.identity.agents.config.load.interval` property are disabled, cached configuration remains valid until the Java agent restarts.

Session Cache

After authentication, AM presents the client with a JWT containing session information. The agent stores part of that session information in the cache. A session stored in the session cache is valid until one of the following events occurs:

- The session contained in the JWT expires.
- The client logs out from AM, and session notifications are enabled.
- The session reaches the expiration time specified by the `org.forgerock.openam.agents.config.active.session.cache.ttl.minutes` property.

Policy Decision Cache

When a client attempts to access a protected resource, the Java agent checks whether there is a policy decision cached for the resource:

- If the client's session is valid, the Java agent requests a policy decision from AM and then enforces it.
- If the client's session is not valid, the Java agent redirects the client to AM for authentication regardless of why the session is invalid. The agent does not specify the reason why the client needs to authenticate.

  Once the client authenticates, the Java agent requests policy decision to AM and enforces it.

Policy decisions are valid in the cache until one of the following events occur:

Session and Policy Validity in Cache

Event	What is invalidated?
Session contained in the JWT expires	Session and policy decisions related to the session

Event	What is invalidated?
Client logs out from AM (and session notifications are enabled)	Session and policy decisions related to the session
Policy decision reaches the expiration time specified by the `com.sun.identity.agents.polling.interval` property	Policy decision
Administrator makes a change to policy configuration (and policy notifications are enabled)	All sessions and all policy decisions

Important

A Java agent that loses connectivity to AM cannot request policy decisions. Therefore, the Java agent denies access to inbound requests that do not have a policy decision cached until the connection is restored.

Not-Enforced Lists Hit and Miss Caches

The first time the Java agent receives a request for a resource, it matches the request and the client's IP address against the rules specified in the not-enforced lists.

Java agents maintain a hit cache and a miss cache for each of the not-enforced lists specified in "Not-Enforced Lists". To speed up future requests, the agent stores whether the resource hit or missed not-enforced rules in the corresponding caches. Therefore, if a request for the same resource reaches the agent again, the agent replays the result of the rules' evaluation stored in the caches instead of re-evaluating the request.

Entries stored in the hit and miss caches do not expire unless AM notifies the agent about configuration changes in the not-enforced lists properties.

For more information about not-enforced cache lists, see "Not-Enforced Lists", Not-Enforced URI Processing Properties, and Not-Enforced IP Processing Properties.

POST Data Preservation Cache

When POST data preservation is enabled, the Java agent caches HTML form data submitted as an HTTP POST by unauthenticated clients.

The POST data expires either when the client recovers the information from the cache or after the time interval specified by the `com.sun.identity.agents.config.postdata.preserve.cache.entry.ttl` property.

For more information about POST data preservation and its properties, see "POST Data Preservation" and POST Data Preservation Properties.

OpenID Connect JSON Web Token (JWT) Cache

Decoding JWTs into JSON objects is a CPU-intensive operation. To reduce the amount of processing required on each request, Java agents cache decoded JWTs.

When a Java agent receives a request for a resource, it passes the JWT through a fast hashing algorithm that creates a 128-bit hash unique for that JWT. Then the agent determines if the hash is in the JWT cache. One of the following scenarios occur:

- The hash is in the cache. The Java agent retrieves the decoded JWT from the cache and continues processing the request.
- The hash is not in the cache. The Java agent decodes the JWT and stores it and its hash in the cache. Then it continues processing the request.

JWTs in the cache expire after the time interval specified by the `org.forgerock.openam.agents.config.jwt.cache.ttl.minutes` property.

For information about the properties that control the JWT cache, see Profile Properties.

Query Parameter Handling

By default, Java agents consider any query parameters to be part of the URL, and insert the entire string into the policy decision cache. For example, the agent will insert each of the following URLs in the cache, even though the root URL is the same:

```
http://agent.example.com:8080/protected/resource.jsp
http://agent.example.com:8080/protected/resource.jsp?a=value1
http://agent.example.com:8080/protected/resource.jsp?b=value2
```

Applications adding new parameters to the URL on every request would fill the Java agent's policy cache without actually using it, which in turn causes the agent to request policy decision to AM each time.

To prevent this behavior, Java agents can be configured to either retain nominated URL parameters (for example, to remove all but those that are added as part of the policy evaluation) or to discard them (for example, to remove all parameters added by the `angular.js` framework).

To retain nominated query parameters, configure one of the following properties:

- Retain Query Parameters (`org.forgerock.openam.agents.config.conditional.wanted.http.url.params`)
- Regular Expression Retain Query Parameters (`org.forgerock.openam.agents.config.conditional.wanted.http.url.params.regexp`)

To remove nominated query parameters, configure one of the following properties:

- Remove Query Parameters (`org.forgerock.openam.agents.config.conditional.unwanted.http.url.params`)
- Regular Expression Remove Query Parameters (`org.forgerock.openam.agents.config.conditional.unwanted.http.url.params.regexp`)

The properties are mutually exclusive and Java agents check them in the following order of precedence:

1. Remove Query Parameters
2. Regular Expression Remove Query Parameters
3. Retain Query Parameters
4. Regular Expression Retain Query Parameters

Warning

Java agents strip the nominated query parameters from the URL *before* taking the following actions:

- Asking AM for policy evaluation
- Checking the not-enforced lists

Ensure the policies defined in AM and the not-enforced rules configured for the agent do not expect a parameter that has been removed.

For more information about these properties, see Query Parameter Handling Properties.

Chapter 2

Preparing for Installation

This chapter covers tasks to perform before installing Java agents in your environment. The following table contains a list of the tasks:

Task	Section
Download Java agent binaries	Section
Secure communications between AM and the Java agents	Section
Create agent profiles	Section
Configure your environment when communication between AM and agents happens behind load balancers or reverse proxies	Section

Downloading and Unzipping Java Agents

Navigate to the ForgeRock BackStage website and choose the agent to download based on your version, architecture, and operating system requirements. Remember to verify the checksum of the downloaded file against the checksum posted on the download page.

Unzip the file in the directory where you plan to store the Java agent's configuration and log files. The following directories are extracted:

`bin`

The **agentadmin** installation and configuration program. For more information about the tool, see "Command-Line Tool Reference"

`config`

Configuration templates used by the **agentadmin** command during installation

`data`

Not used

`etc`

Configuration templates used during installation

`installer-logs`

Location of log files written during installation

`legal-notices`

Licensing information including third-party licenses

`lib`

Shared libraries used by the Java agent

`locale`

Property files used by the installation program

`README`

README file containing platform and install information for the agent

Configuring Access Management Servers to Communicate With Java Agents

AM communicates all authentication and authorization information to Java agents using OpenID Connect (OIDC) JSON web tokens (JWT). To secure the integrity of the JSON payload (outlined in the JSON Web Algorithm specification RFC 7518), AM and the Java agent support signing the tokens for communication with the RS256 algorithm.

AM also uses an HMAC signing key to protect requested `ACR` claims values between sending the user to the authentication endpoint, and returning from successful authentication.

By default, AM uses a demo key and an autogenerated secret for these purposes. For production environments, perform the steps in one of the following procedures to create new key aliases and configure them in AM:

- "To Configure Access Management Secret IDs for the Agents' OAuth 2.0 Provider in AM 6.0 or earlier"
- "To Configure Access Management Secret IDs for the Agents' OAuth 2.0 Provider in AM 6.5 or later"

To Configure Access Management Secret IDs for the Agents' OAuth 2.0 Provider in AM 6.0 or earlier

By default, AM 6.0 or earlier signs the JWTs with the `test` key alias provided in AM's JCEKS keystore and sign the claims with a secret autogenerated at time.

Perform the following steps to create and set up a new key and a new secret in AM 6.0 or earlier:

1. Create the following aliases in one of the secret stores configured in AM, for example, the default JCEKS keystore:

   a. Create an RSA key pair.

      For more information about creating a key alias in the AM keystore, see the section Creating Key Aliases of the *ForgeRock Access Management Setup and Maintenance Guide*.

   b. Create an HMAC secret.

2. In the AM console, navigate to Configure > Global Services > OAuth2 Provider.

3. Perform the following actions:

   a. Replace the `test` key alias in the ID Token Signing Key Alias for Agent Clients field with the new RSA key alias.

   b. Replace the value in the Authenticity Secret field with the new HMAC secret.

      Note that you may already have a secret configured for this secret ID, since it is also used for signing certain OpenID Connect ID tokens and remote consent requests.

   c. Save your changes.

   No further configuration is required in the agents.

To Configure Access Management Secret IDs for the Agents' OAuth 2.0 Provider in AM 6.5 or later

By default, AM 6.5 or later is configured to:

- Sign the JWTs with the secret mapped to the `am.global.services.oauth2.oidc.agent.idtoken.signing` secret ID. This secret ID defaults to the `rsajwtsigningkey` key alias provided in AM's JCEKS keystore.
- Sign the claims with the secret mapped to the `am.services.oauth2.jwt.authenticity.signing` secret ID. This secret ID defaults to the `hmacsigningtest` key alias available in AM's JCEKS keystore.

Perform the following steps to create and set up new keys on a keystore secret store:

1. Create the following aliases in one of the secret stores configured in AM, for example, the default JCEKS keystore:

   a. Create an RSA key pair.

   b. Create an HMAC secret.

2. In the AM console, navigate to Configure > Secret Stores > *Keystore Secret Store Name* > Mappings.

3. Configure the following secret IDs:

a. Configure the new RSA key alias in the `am.global.services.oauth2.oidc.agent.idtoken.signing` secret ID.

b. Configure the new HMAC secret in the `am.services.oauth2.jwt.authenticity.signing` secret ID.

   Note that you may already have a secret configured for this secret ID, since it is also used for signing certain OpenID Connect ID tokens and remote consent requests. For more information, see Secret ID Mapping Defaults in the *ForgeRock Access Management Setup and Maintenance Guide*.

c. Save your changes.

For more information about secret stores, see the chapter Setting Up Secret Stores of the *ForgeRock Access Management Setup and Maintenance Guide*.

No further configuration is required in the agents.

Creating Agent Profiles

A Java agent requires a profile to connect to and communicate with AM, regardless of whether it is stored centrally in AM or on the agent installation.

To Create an Agent Profile in AM Using the Console

Create an agent profile using the AM console by performing the following steps:

1. In the AM console, navigate to Realms > *Realm Name* > Applications > Agents > Java, and then select the `Add Java Agent` button in the Agent tab.

2. Complete the web form using the following hints:

   Agent ID

   The ID of the agent profile. This ID is used during the agent installation.

   Agent URL

   The URL the Java agent protects, such as `http://www.example.com:8080/agentapp`.

   In centralized configuration mode, the Agent URL is used to populate the agent profile for services, such as notifications.

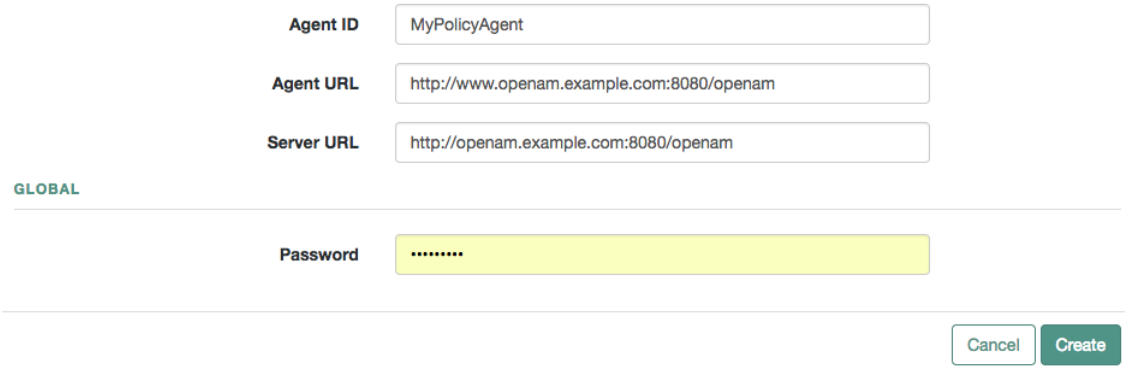
   Server URL

   The full URL to an AM instance. If AM is deployed in a site configuration (behind a load balancer), enter the site URL.

In centralized configuration mode, Server URL is used to populate the agent profile for use with as login, logout, naming, and cross-domain SSO.

Password

The password the agent uses to authenticate to AM. Use this password when installing an agent.



To Create an Agent Profile Using the ssoadm Command

You can create an agent profile in AM using the **ssoadm** command-line tool. You do so by specifying the agent properties either as a list of attributes, or by using an agent properties file as shown below.

Perform the following steps to create an agent profile using the **ssoadm** command:

1. Make sure the **ssoadm** command is installed. See the section Installing and Using the Tools in the *ForgeRock Access Management Install Guide*.

2. Create a password file, for example `$HOME/.pwd.txt`. The file should only contain the password string, on a single line.

   The password file must be read-only for the user who creates the agent profile, and must not be accessible to other users:

   ```
   $ chmod 400 $HOME/.pwd.txt
   ```

3. Create the agent profile, specifying `--agenttype J2EEAgent`:

```
$ ssoadm create-agent \
 --realm / \
 --agentname MyJavaEEAgent \
 --agenttype J2EEAgent \
 --adminid amadmin \
 --password-file $HOME/.pwd.txt \
 --datafile MyJavaEEAgent.properties

Agent configuration was created.
```

4. Review the new profile in the AM console under Realms > *Realm Name* > Applications > Agents > Java > *Agent Name*.

To Create an Agent Profile Group and Inherit Settings

Agent profile groups let you set up multiple agents that inherit settings from the group. To create a new agent profile group, perform the following steps:

1. In the AM console, navigate to Realms > *Realm Name* > Applications > Agents > Java.
2. Select New in the Group table, and provide a name for the group and the URL to the AM server in which to store the profile.

   After creating the group profile, you can select the link to the new group profile to fine-tune the configuration.
3. Inherit group settings by selecting your agent profile, and then selecting the group name in the Group drop-down list near the top of the profile page.

   You can then adjust inheritance by clicking Inheritance Settings on the AM Services agent profile tab.

Delegating Agent Profile Creation

If you want to create agent profiles when installing Java agents, then you need the credentials of an AM user who can read and write agent profiles.

You can use the AM administrator account when creating agent profiles. If you delegate agent installation, then you might not want to share AM administrator credentials with everyone who installs Java agents.

To Create Agent Administrators for a Realm

Follow these steps to create *agent administrator* users for a realm:

1. In the AM console, navigate to Realms > *Realm Name* > Subjects.
2. Under Group click New... and create a group for agent administrators.

3. Switch to the Privileges tab for the realm, and click the name of the group you created.
4. Select Read and write access to all configured agents, and then Save your work.
5. Return to the Subjects tab, and under User create as many agent administrator users as needed.
6. For each agent administrator user, edit the user profile.

   Under the Group tab of the user profile, add the user to agent profile administrator group, and then Save your work.
7. Provide each system administrator who installs Java agents with their agent administrator credentials.

   When installing the Java agent with the `--custom-install` option, the system administrator can choose the option to create the profile during installation, and then provide the agent administrator user name and the path to a read-only file containing the agent administrator password. For silent installs, you can add the `--acceptLicense` option to auto-accept the software license agreement.

Supporting Load Balancers and Reverse Proxies Between AM and the Agents

When your environment has reverse proxies or load balancers configured between the agents and AM, you must perform additional configuration in both AM and your environment before installing the agents.

Failure to do so may cause the agent installation to fail, or it may compromise the agent's functionality.

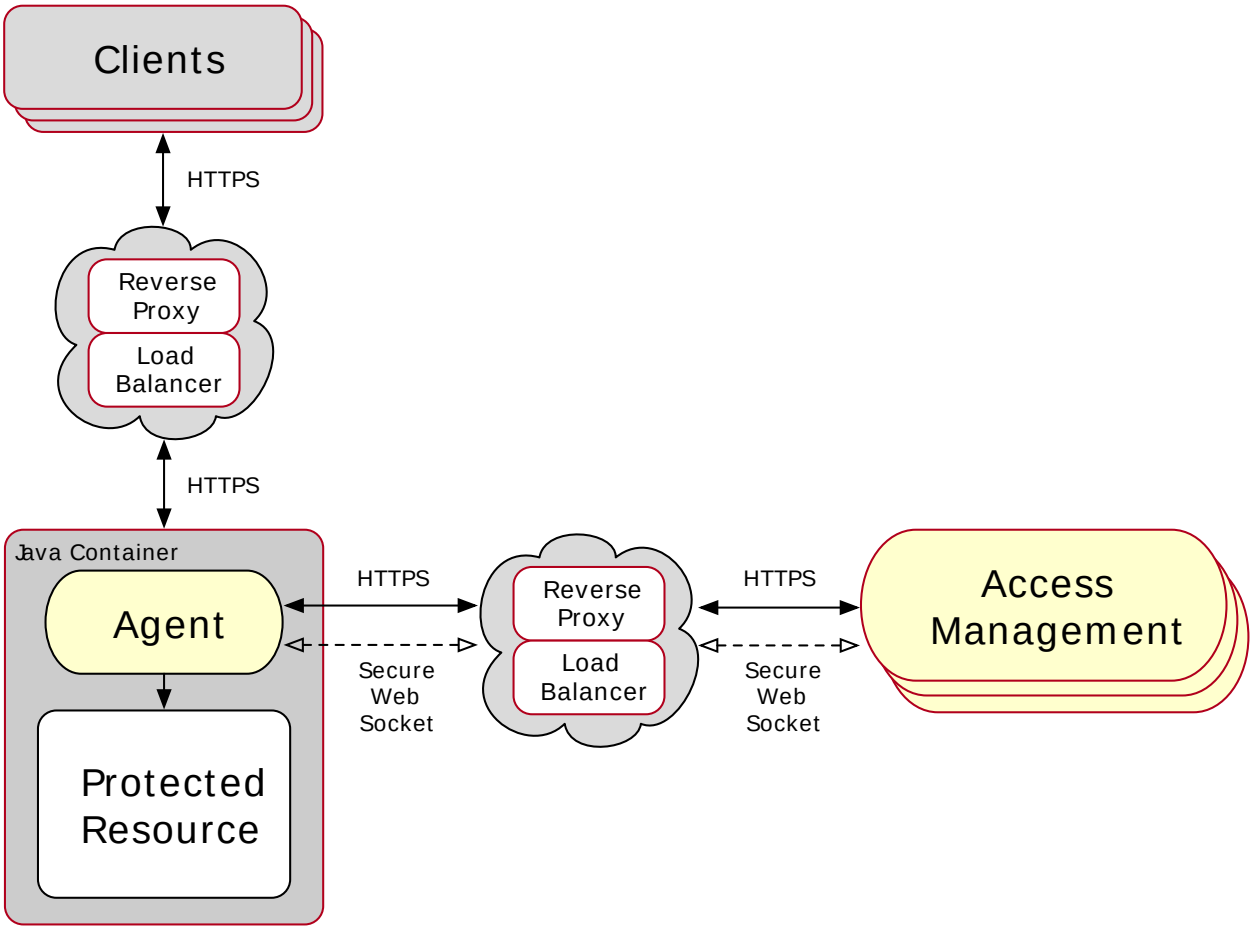
For more information, see "*Configuring Environments With Load Balancers and Reverse Proxies*".

Chapter 3

Configuring Environments With Load Balancers and Reverse Proxies

When working with AM and agents, the most common deployment scenario is to configure a load balancer and a reverse proxy between the clients and the agents, and another load balancer and reverse proxy between the agent and an AM site, as shown in the following diagram:

Java Agents in Environments with Load Balancers and Reverse Proxies



Usually, you want to anonymize client traffic as it gets into your network by using a reverse proxy, then balance the load among different application servers and agents.

AM sites are usually deployed behind a load balancer so the load can be spread among different instances. A reverse proxy may be deployed in front of the AM site to protect its APIs, too.

Note that the reverse proxy and the load balancer may be the same entity. In very complex environments, there may be more than the depicted load balancers and reverse proxies deployed in the network.

In any case, when installing Java agents in an environment with load balancers or reverse proxies, you must consider the communication between the clients and the Java agents, and between the agents and the AM servers.

Refer to the following sections for more information:

- "Regarding Communication Between AM and Agents".
- "Regarding Communication Between Clients and Agents".

Regarding Communication Between AM and Agents

Before attempting to install Java agents in an environment where AM is behind a load balancer, reverse proxy, or both, consider the following points:

Agent's IP Address and/or FQDN

When a load balancer or a reverse proxy is configured between AM and the Java agents, the agents' IP addresses and FQDNs are concealed by the load balancer/reverse proxy's own IP or FQDN. As a result, AM cannot determine the agents' base URL as expected.

This could cause trouble during the installation process and also hinder functionality such as redirection using the `goto` parameter.

Therefore, you must configure the following:

- The load balancer or reverse proxy, to forward the agents' IP address and/or FQDN in a header.
- The AM site, to recover the forwarded headers. For more information, see "Configuring AM to Use Forwarded Headers".

> **Note**
>
> A load balancer or reverse proxy conceals the AM instances' IP addresses and FQDNs. When installing Java agents, use the load balancer or reverse proxy IP address or FQDN as the point of contact for the AM site.

AM Sessions and Session Stickiness

When Java agents communicate with an AM site that is behind a load balancer, you can improve policy evaluation performance by setting up AM's sticky cookie (by default, `amlbcookie`) to the AM's server ID. For more information, see Configuring Site Sticky Load Balancing in the *ForgeRock Access Management Installation Guide*.

> **Important**
>
> When configuring multiple agents behind a load balancer or reverse proxy, you must take into consideration whether you use one or multiple agent profiles, since it impacts sticky load balancer requirements:

- If the agents are configured with multiple agent profiles you must configure sticky load balancing. This is because the agent profile name is contained in the OpenID Connect JWT the agent and AM use to communicate. Without session stickiness, there is no way to make sure that the appropriate JWT ends in the appropriate Java agent instance.
- If multiple agents are configured with the same agent profile, you can decide whether to configure sticky load balancing or not depending on other requirements of your environment.

WebSockets

Your load balancers and reverse proxies must support the WebSocket protocol for communication between the Java agents and the AM servers.

For more information, refer to the load balancer or proxy documentation.

Tip

For an example of how to configure Apache HTTP as a reverse proxy, see "Configuring Apache HTTP Server as a Reverse Proxy Example".

Configuring AM to Use Forwarded Headers

When Java agents are behind a load balancer or reverse proxy, you must configure AM to recover the forwarded headers that expose the agents' real IP address or FQDN.

To Configure Access Management to Use Forwarded Headers

To configure how AM obtains the base URL of Java agents, use the Base URL Source service:

1. Log in to the AM console as an administrative user, such as `amAdmin`.
2. Navigate to Realms > *Realm Name* > Services.
3. Select Add a Service, select Base URL Source, and then select Create, leaving the fields empty.
4. Configure the service with the following properties:

   - **Base URL Source**: X-Forwarded-* headers

     This property allows AM to retrieve the base URL from the `Forwarded` header field in the HTTP request. The Forwarded HTTP header field is standardized and specified in *RFC 7239*.

   - **Context path**: *AM's deployment uri*. For example, `/openam`.

   Leave the rest of the fields empty.

Tip

For more information about the Base URL Source service, see Base URL Source in the *ForgeRock Access Management Reference*.

5. Save your changes.

Regarding Communication Between Clients and Agents

When your environment has load balancers or reverse proxies between clients and agents, you must consider the following points:

Client's IP Address and/or FQDNs

When configuring Java agents behind a load balancer or reverse proxy, the clients' IP addresses and FQDNs are hidden by the load balancer's IP or FQDN, which results in agents not being able to determine the clients' base URLs.

Therefore, you must configure the load balancer or reverse proxy to forward the client's IP address and/or the client's FQDN in a header. Failure to do so will will prevent the agent from performing policy evaluation, and applying not-enforced and conditional login/logout rules.

For more information, see "Configuring Client Identification Properties".

POST Data Preservation

When using POST data preservation, you must use sticky load balancing to ensure that the client always hits the same agent and, therefore, their saved POST data.

Java agents provide properties to set either a sticky cookie or a URL query string for load balancers and reverse proxies.

For more information, see "Configuring POST Data Preservation for Load Balancers or Reverse Proxies".

Java Containers FQDNs, Ports, and Protocols

When the protected Java containers and their agents are behind a load balancer or reverse proxy, it is imperative that the agent is configured to match the load balancer FQDN, port, and protocol.

Failure to do so would make the agent to return HTTP 403 errors when clients request access to resources.

There are two use-cases:

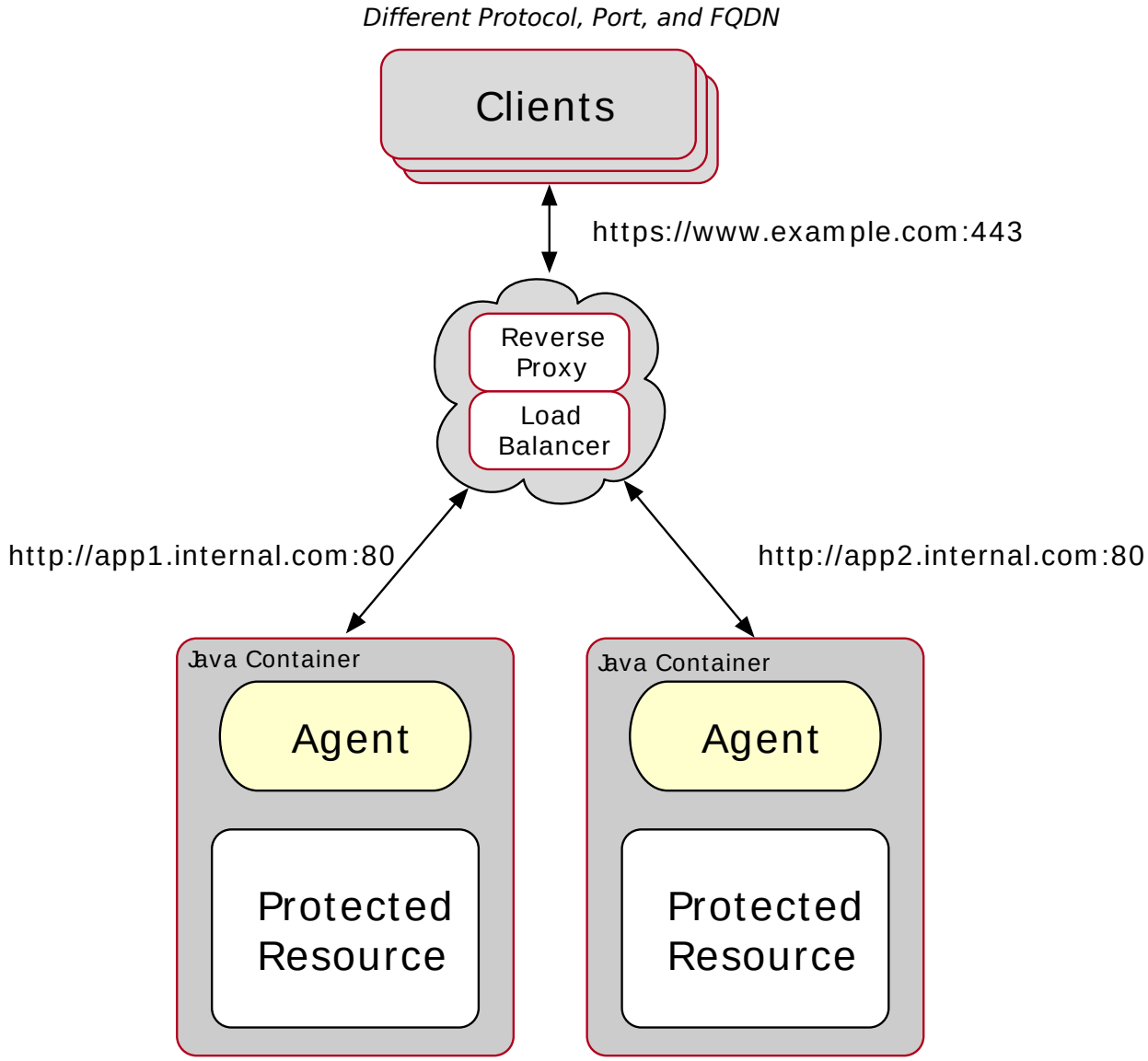
- The load balancer or reverse proxy forwards requests and responses between clients and protected Java containers only. In this case, ports and protocols configured in the Java container match those on the load balancer or reverse proxy, but FQDNs do not.
- The load balancer or reverse proxy also performs SSL offloading, terminating the SSL traffic and converting the requests reaching the Java container to HTTP. This reduces the load on the protected containers, since the processing of the public key is usually done by a hardware accelerator.

  In this case, neither ports, protocols, or FQDNs match.

For more information about matching FQDNs, ports and protocols, see "Matching Protected Java Container Ports, Protocols, and FQDNs".

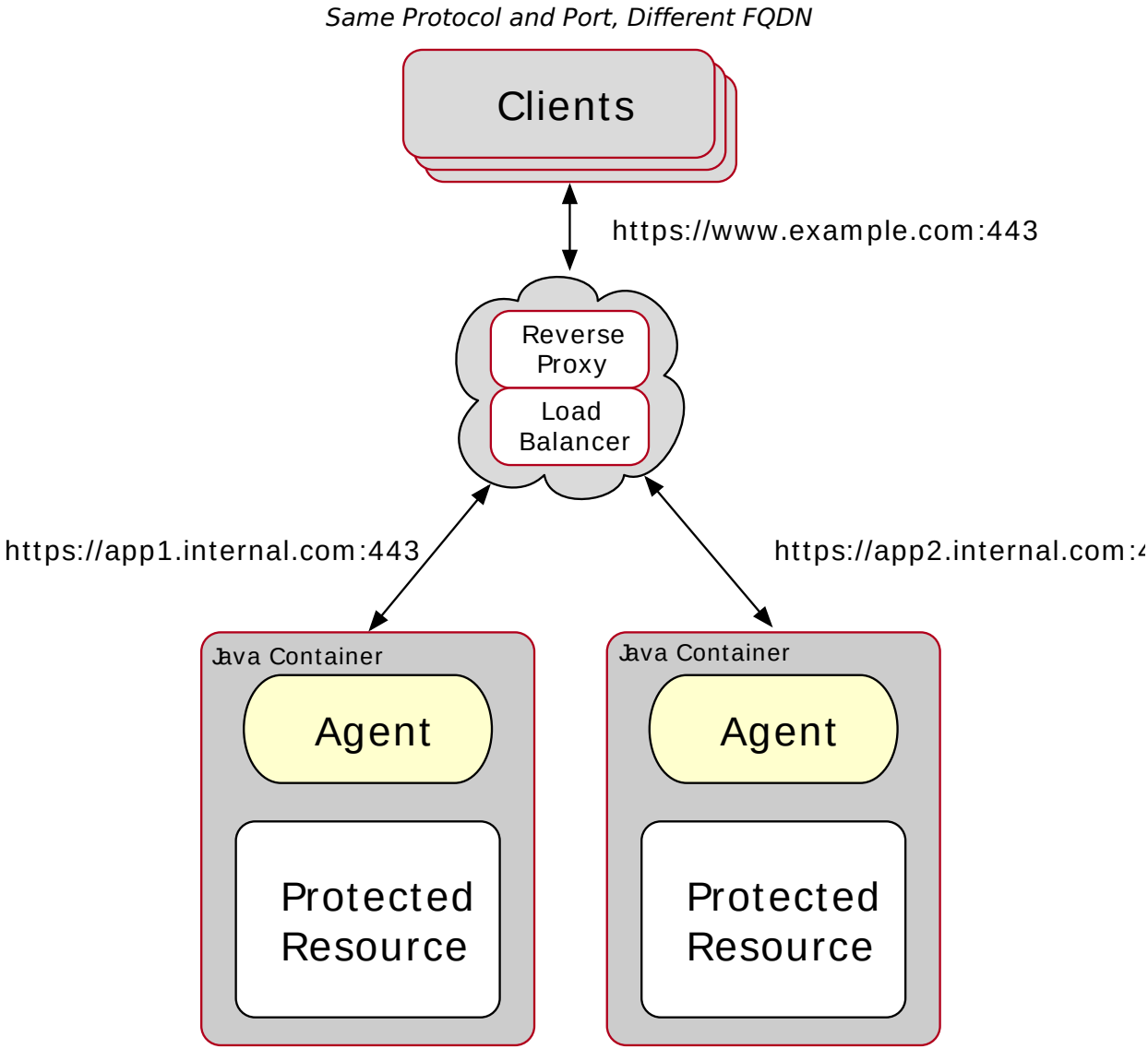
Matching Protected Java Container Ports, Protocols, and FQDNs

When the protocol and port configured on the load balancer or reverse proxy differ from those configured on the protected Java container, you must override them in the Java agent configuration. The following diagram illustrates this scenario:



In this case, configure the Java agents following the steps in "To Override Protocol, Host, and Port".

When the protocol and port configured on the load balancer or reverse proxy match those configured on the protected Java container, you must map the agent host name to the load balancer or reverse proxy host name. The following diagram illustrates this scenario:



In this case, configure the Java agents following the steps in "To Map the Agent Host Name to the Load Balancer or Reverse Proxy Host Name".

To Override Protocol, Host, and Port

Use the alternate Java agent URL properties to override the agent protocol, host, and port with that of the load balancer or reverse proxy.

Important

The Java agent configuration for SSL offloading has the side effect of preventing FQDN checking and mapping. As a result, URL rewriting and redirection does not work correctly when the Java agent is accessed directly and not through the load balancer or proxy. This should not be a problem for client traffic, but potentially could be an issue for applications accessing the protected container directly, from behind the load balancer.

This procedure explains how to do so for a centralized Java agent profile configured in the AM console. The steps also mention the properties for Java agent profiles that rely on local, file-based configurations:

1. Log in to the AM console as an administrative user with rights to modify the Java agent profile.
2. Navigate to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced.
3. Set the Alternative Agent Host Name to that of the load balancer or reverse proxy. For example, `lb.example.com`.

   The equivalent property setting is `com.sun.identity.agents.config.agent.host=lb.example.com`.
4. Set the Alternative Agent Port number to that of the load balancer or proxy. For example, `80`.

   The equivalent property setting is `com.sun.identity.agents.config.agent.port=80`.
5. Set the Alternative Agent Protocol to that of the load balancer or proxy. For example, `http` or `https`.

   The equivalent property setting is `com.sun.identity.agents.config.agent.protocol=https`.
6. Save your work.
7. Restart the Java container where the agent is installed.

To Map the Agent Host Name to the Load Balancer or Reverse Proxy Host Name

When protocols and port numbers match, configure fully qualified domain name (FQDN) mapping.

This procedure explains how to do so for a centralized Java agent profile configured in the AM console. The steps also mention the properties for Java agent profiles that rely on local, file-based configurations:

1. Log in to the AM console as an administrative user with rights to modify the Java agent profile.

2. Navigate to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name*.
3. In the Global tab, enable FQDN Check.

   The equivalent property setting is `com.sun.identity.agents.config.fqdn.check.enable=true`.
4. Set the FQDN Default field to the fully qualified domain name of the load balancer or proxy, such as `lb.example.com`, rather than the protected container FQDN where the Java agent is installed.

   The equivalent property setting is `com.sun.identity.agents.config.fqdn.default=lb.example.com`.
5. Append the FQDN of the load balancer or proxy to the Agent Root URL for CDSSO field.

   The equivalent property setting is `sunIdentityServerDeviceKeyValue[n]=lb.example.com`.
6. Map the load balancer or proxy FQDN to the FQDN where the Java agent is installed in the FQDN Virtual Host Map key-pair map. For example, set the key `agent.example.com` (protected Java container) and a value `lb.example.com` (load balancer or proxy).

   The equivalent property setting is `com.sun.identity.agents.config.fqdn.mapping[agent.example.com]=lb.example.com`.
7. Save your work.
8. Restart the Java container where the agent is installed.

Configuring Client Identification Properties

After configuring your proxies or load balancers to forward the client's FQDN and/or IP address, configure the Java agents to check the appropriate headers.

To Configure the Java Agent Client Identification Properties

This procedure explains how to configure the client identification properties for a centralized Java agent profile configured in the AM console. The steps also mention the properties for Java agent profiles that rely on local, file-based configurations:

1. Log in to the AM console with a user that has permissions to modify the Java agent profile.
2. Navigate to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced.
3. (Optional) In the Client IP Address Header field, configure the name of the header containing the IP address of the client. For example, `X-Forwarded-For`.

   Configure this property if your AM policies are IP address-based, you configured the agent for not-enforced IP rules, or if you configured the agent to take any decision based on the client's IP address.

   The equivalent property setting is `com.sun.identity.agents.config.client.ip.header=X-Forwarded-For`.

4. (Optional) In the Client Hostname Header field, configure the name of the header containing the FQDN of the client. For example, `X-Forwarded-Host`.

   Configure this property if your AM policies are URL-based, you configured the agent for not-enforced URL rules, or if you configured the agent to take any decision based on the client's URL.

   The equivalent property setting is `com.sun.identity.agents.config.client.hostname.header=X-Forwarded-Host`.

5. Save your changes.

Configuring POST Data Preservation for Load Balancers or Reverse Proxies

When configuring POST data preservation behind a load balancer or a reverse proxy, you must configure both your load balancer/reverse proxy and the Java agents for session stickiness.

This procedure explains how to configure the client identification properties for a centralized Java agent profile configured in the AM console. The steps also mention the properties for Java agent profiles that rely on local, file-based configurations:

To Configure POST Data Preservation Stickiness Properties

1. Log in to the AM console with a user that has permissions to modify the Java agent profile.
2. Navigate to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced.
3. Decide whether the Java agent should create a cookie or append a string to the URL to assist with sticky load balancing.

   In the PDP Stickysession mode drop-down menu, configure one of the following options:

   - **Cookie**. The Java agent will create a cookie for POST data preservation session stickiness. The contents of the cookie is configured in the next step.
   - **URL**. The Java agent will append to the URL a string specified in the next step.

   The equivalent property setting is `com.sun.identity.agents.config.postdata.preserve.stickysession.mode=[Cookie|URL]`.

4. In the PDP Stickysession key-value field, configure a key-pair value separated by the **=** character.

   For example, specifying `lb=myserver` either sets a cookie called `lb` with `myserver` as a value, or appends `lb=myserver` to the URL query string.

   The equivalent property setting is `com.sun.identity.agents.config.postdata.preserve.stickysession.value=lb=myserver`.

5. Save your changes.

6. Configure your load balancer or reverse proxy to ensure session stickiness when the cookie or URL query parameter are present.

Chapter 4

Installing Java Agents

Install Java agents in web application containers to police access to your web sites, web applications, and resources. Java agents depend on AM for all authentication and authorization decisions. The primary responsibility of Java agents is to enforce what AM decides in a way that is unobtrusive to the user.

When installing Java agents, consider the following points:

- Configurations where AM and the Java agent are installed in the same container are not supported.
- A single Java agent installation can hold multiple agent instances. Therefore, install only one Java agent per application server and configure as many agent instances as you require. Installing more than one Java agent in an application server is not supported.

The following table contains a list of sections containing information to install Java agents on supported platforms:

Task	Section
Install Java agents on Apache Tomcat	Section
Install Java agents on Red Hat JBoss	Section
Install Java agents on Eclipse Jetty	Section
Install Java agents on Oracle WebLogic	Section
Install Java agents on IBM WebSphere	Section

Installing the Tomcat Java Agent

This section covers prerequisites and installation procedures for Java Agents 5.6 on Tomcat.

Before You Install

1. Download the agent from BackStage. For more information, see "Downloading and Unzipping Java Agents".
2. Consider the following points before installing the Tomcat Java agent:
   - Install Tomcat before you install the agent.

- All of the Tomcat scripts must be present in the `$CATALINA_HOME/bin` directory. The Tomcat Windows executable installer does not include the scripts, for example. If the scripts are not present in your installation, copy the contents of the `bin` directory from a `.zip` download of Tomcat of the same version as the one you installed.
- Install a supported version of the Java runtime environment, as described in "Java Requirements" in the *Release Notes*. Set the `JAVA_HOME` environment variable accordingly. The agent installer requires Java.

```
$ echo $JAVA_HOME
/path/to/java
```

Installing the Tomcat Java Agent

Complete the following procedures to install the Tomcat Java Agent:

To Complete Pre-Installation Tasks

Perform the following steps to create the configuration required by the Java agent before installing it:

1. Create at least one policy in AM to protect resources with the agent, as described in the procedure *Implementing Authorization Using the Access Management Console*.
2. Create an agent profile in AM, required by the Java agent to connect and communicate with AM. For more information, see "Creating Agent Profiles".
3. Ensure that the key pair configured for signing the OpenID Connect JWTs exchanged between AM and the Java agents is not the default `test` key pair. For more information, see "Configuring Access Management Servers to Communicate With Java Agents".
4. Configure AM to protect the cross-domain single sign-on (CDSSO) session cookie from hijacking. For more information, see Implementing Cross-Domain Single Sign-On in the *ForgeRock Access Management Authentication and Single Sign-On Guide*.
5. Consider the communication between the agents and the AM servers, and between the agents and the clients when installing agents in environments with load balancers and/or reverse proxies. For more information, see "*Configuring Environments With Load Balancers and Reverse Proxies*".
6. Create a text file containing only the password specified when creating the agent profile, and protect it:

   Windows example:

   ```
   C:\> echo password > pwd.txt
   ```

   In Windows Explorer, right-click the password file, for example `pwd.txt`, select Read-Only, and then click OK.

UNIX example:

```
$ echo password > /tmp/pwd.txt
```

```
$ chmod 400 /tmp/pwd.txt
```

To Install the Tomcat Java Agent

1. Shut down the Tomcat server where you plan to install the agent.
2. Make sure AM is running.
3. Run **agentadmin --install** to install the agent:

   ```
   $ /path/to/java_agents/tomcat_agent/bin/agentadmin --install --acceptLicense
   ```

   a. When you run the command, you will be prompted to read and accept the software license agreement for the agent installation. You can suppress the license agreement prompt by including the `--acceptLicence` parameter. The inclusion of the option indicates that you have read and accepted the terms stated in the license. To view the license agreement, open `<server-root>/legal-notices/license.txt`.

   b. Enter the path to the Tomcat configuration folder. For example, `/path/to/apache-tomcat/conf`.

   ```
   Enter the complete path to the directory which is used by Tomcat Server to
   store its configuration Files. This directory uniquely identifies the
   Tomcat Server instance that is secured by this Agent.
   [ ? : Help, ! : Exit ]
   Enter the Tomcat Server Config Directory Path
   [/opt/apache-tomcat-6.0.14/conf]: /path/to/apache-tomcat/conf
   ```

   c. Enter the AM URL. For example, `https://openam.example.com:8443/openam`.

   To balance agent connections to an AM site, configure the URL of the load balancer in front of the AM site.

   Note

   If your environment has a reverse proxy configured between AM and the agent, set the AM URL to the proxy URL instead. For example, `https://proxy.example.com:443/openam`. For more information about setting up the environment for reverse proxies, see "Configuring Apache HTTP Server as a Reverse Proxy Example".

   ```
   Enter the URL where the AM server is running. Please include the
   deployment URI also as shown below:
   (http://openam.sample.com:58080/openam)
   [ ? : Help, < : Back, ! : Exit ]
   AM server URL: https://openam.example.com:8443/openam
   ```

   d. Enter the `$CATALINA_HOME` environment variable specifying the path to the root of the Tomcat server. For example, `/path/to/apache-tomcat`.

```
$CATALINA_HOME environment variable is the root of the tomcat
installation.
[ ? : Help, < : Back, ! : Exit ]
Enter the $CATALINA_HOME environment variable: /path/to/apache-tomcat
```

e. Enter the agent URL. For example, `http://www.example.com:8080/agentapp`.

```
Enter the Agent URL. Please include the deployment URI also as shown below:
(http://agent1.sample.com:1234/agentapp)
[ ? : Help, < : Back, ! : Exit ]
Agent URL: http://www.example.com:8080/agentapp
```

f. Enter the agent profile name that you created in AM as part of the pre-installation procedure. For example, `TomcatAgent`.

```
Enter the Agent profile name
[ ? : Help, < : Back, ! : Exit ]
Enter the Agent Profile name: TomcatAgent
```

g. Enter the realm in which the specified agent profile exists.

Press **ENTER** to accept the default value of `/`, signifying the top-level realm.

> **Note**
>
> If you specify the Accept Empty value (^) option, the top-level realm is assumed.

```
Enter the Agent profile realm
[ ? : Help, < : Back, ! : Exit, ^ : Accept Empty value ]
Enter the Agent Profile realm [/]:
```

h. Enter the path to the password file you created as part of the pre-installation procedure. For example, `/tmp/pwd.txt`.

```
Enter the path to a file that contains the password to be used for identifying
the Agent.
[ ? : Help, < : Back, ! : Exit ]
Enter the path to the password file: /tmp/pwd.txt
```

4. Review a summary of your responses and select an action to continue: install, go back a step, start over, or exit from the install:

```
---------------------------------------------
SUMMARY OF YOUR RESPONSES
---------------------------------------------
Tomcat Server Config Directory : /path/to/tomcat/conf

AM server URL : https://openam.example.com:8443/openam
$CATALINA_HOME environment variable : /path/to/tomcat

Agent URL : http://www.example.com:8080/agentapp
Agent Profile name : TomcatAgent
Agent Profile Realm : /
Agent Profile Password file name : /tmp/pwd.txt
```

```
Verify your settings above and decide from the choices below.
1. Continue with Installation
2. Back to the last interaction
3. Start Over
4. Exit
Please make your selection [1]: 1

...

SUMMARY OF AGENT INSTALLATION
-----------------------------
Agent instance name: Agent_001
Agent Bootstrap file location:
/path/to/java_agents/tomcat_agent/Agent_001/config/
OpenSSOAgentBootstrap.properties
Agent Configuration file location
/path/to/java_agents/tomcat_agent/Agent_001/config/
OpenSSOAgentConfiguration.properties
Agent Audit directory location:
/path/to/java_agents/tomcat_agent/Agent_001/logs/audit
Agent Debug directory location:
/path/to/java_agents/tomcat_agent/Agent_001/logs/debug

Install log file location:
/path/to/java_agents/tomcat_agent/installer-logs/audit/install.log

Thank you for using AM Policy Agent
```

Upon successful completion, the installer adds the agent configuration to the Tomcat configuration, and also set up the configuration and log directories for the agent.

5. Take note of the configuration files and log locations.

   Each agent instance that you install on the system has its own numbered configuration and logs directory. The first agent's configuration and logs are thus located under the directory `java_agents/tomcat_agent/Agent_001/`:

   `config/OpenSSOAgentBootstrap.properties`

   Used to bootstrap the agent, allowing it to connect to AM and download its configuration.

   `config/OpenSSOAgentConfiguration.properties`

   Only used if you configured the agent to use local configuration.

   `logs/audit/`

   Operational audit log directory, only used if remote logging to AM is disabled.

   `logs/debug/`

   Debug directory where the `debug.out` debug file resides. Useful in troubleshooting agent issues.

6. Review Tomcat's global `web.xml` file and your application's `web.xml` files and configure the agent filter.

7. (Optional) If you have a policy configured, you can test your agent installation. For example, try to browse to a resource that your agent protects. You should be redirected to AM to authenticate, for example, as user `demo`, password `changeit`. After you authenticate, AM then redirects you back to the resource you tried to access.

Installing the Tomcat Java Agent Silently

To install the Tomcat Java agent silently you must create a response file containing the installation parameters, and then provide it to the **agentadmin** command.

The following is an example of the response file:

```
# Agent User Response File
CONFIG_DIR= /usr/local/apache-tomcat-9.0.11/conf
AM_SERVER_URL= https://openam.example.com:8443/openam
CATALINA_HOME= /usr/local/apache-tomcat-9.0.11
AGENT_URL= http://www.example.com:8080/agentapp
AGENT_PROFILE_NAME= TomcatAgent
AGENT_PROFILE_REALM= /
AGENT_PASSWORD_FILE= /tmp/pwd.txt
```

To balance agent connections to an AM site, set the `AM_SERVER_URL` variable as the URL of the load balancer in front of the AM site.

Note

If your environment has a reverse proxy configured between AM and the agent, set the AM URL to the proxy URL instead. For example, `https://proxy.example.com:443/openam`. For more information about setting up the environment for reverse proxies, see "Configuring Apache HTTP Server as a Reverse Proxy Example".

You can also create this file automatically when installing the agent by running the **agentadmin** command with the `--saveResponse` option. For example:

```
$ agentadmin --install --saveResponse response-file
```

Complete the following procedures to install the Tomcat Java agent silently:

To Complete Pre-Installation Tasks

Perform the following steps to create the configuration required by the Java agent before installing it:

1. Create at least one policy in AM to protect resources with the agent, as described in the procedure *Implementing Authorization Using the Access Management Console*.

2. Create an agent profile in AM, required by the Java agent to connect and communicate with AM. For more information, see "Creating Agent Profiles".

3. Ensure that the key pair configured for signing the OpenID Connect JWTs exchanged between AM and the Java agents is not the default `test` key pair. For more information, see "Configuring Access Management Servers to Communicate With Java Agents".

4. Configure AM to protect the cross-domain single sign-on (CDSSO) session cookie from hijacking. For more information, see Implementing Cross-Domain Single Sign-On in the *ForgeRock Access Management Authentication and Single Sign-On Guide*.

5. Consider the communication between the agents and the AM servers, and between the agents and the clients when installing agents in environments with load balancers and/or reverse proxies. For more information, see "*Configuring Environments With Load Balancers and Reverse Proxies*".

6. Create a text file containing only the password specified when creating the agent profile, and protect it:

   Windows example:

   ```
   C:\> echo password > pwd.txt
   ```

   In Windows Explorer, right-click the password file, for example `pwd.txt`, select Read-Only, and then click OK.

   UNIX example:

   ```
   $ echo password > /tmp/pwd.txt
   ```

   ```
   $ chmod 400 /tmp/pwd.txt
   ```

To Install the Tomcat Java Agent Silently

1. Review the information in "Before You Install".

2. Shut down the Tomcat server where you plan to install the agent.

3. Make sure that AM is running.

4. Make sure you have a response file ready. For example, `response-file`. For more information, see "Installing the Tomcat Java Agent Silently".

5. Run the **agentadmin** command with the `--useResponse` option. For example:

   ```
   $ agentadmin --install --acceptLicense --useResponse response-file
   ```

6. Review Tomcat's global `web.xml` file and your application's `web.xml` files and configure the agent filter.

Installing the JBoss Java Agent

This section covers prerequisites and installation procedures for Java Agents 5.6 on JBoss. All the examples assume that you are using the agent on JBoss, but the procedures are the same for WildFly.

Before You Install

1. Download the agent from BackStage. For more information, see "Downloading and Unzipping Java Agents".

   Agent binaries for JBoss and WildFly are the same.

2. Consider the following points before installing JBoss Java agents:

   - Install JBoss before installing the agent.
   - Install a supported version of the Java runtime environment, as described in "Java Requirements" in the *Release Notes*. Set the `JAVA_HOME` environment variable accordingly. The agent installer requires Java.

     ```
     $ echo $JAVA_HOME
     /path/to/java
     ```

Installing the JBoss Java Agent

Complete the following procedures to install the JBoss Java agent:

To Complete Pre-Installation Tasks

Perform the following steps to create the configuration required by the Java agent before installing it:

1. Create at least one policy in AM to protect resources with the agent, as described in the procedure *Implementing Authorization Using the Access Management Console*.

2. Create an agent profile in AM, required by the Java agent to connect and communicate with AM. For more information, see "Creating Agent Profiles".

3. Ensure that the key pair configured for signing the OpenID Connect JWTs exchanged between AM and the Java agents is not the default `test` key pair. For more information, see "Configuring Access Management Servers to Communicate With Java Agents".

4. Configure AM to protect the cross-domain single sign-on (CDSSO) session cookie from hijacking. For more information, see Implementing Cross-Domain Single Sign-On in the *ForgeRock Access Management Authentication and Single Sign-On Guide*.

5. Consider the communication between the agents and the AM servers, and between the agents and the clients when installing agents in environments with load balancers and/or reverse

proxies. For more information, see "*Configuring Environments With Load Balancers and Reverse Proxies*".

6. Create a text file containing only the password specified when creating the agent profile, and protect it:

   Windows example:

   ```
   C:\> echo password > pwd.txt
   ```

   In Windows Explorer, right-click the password file, for example `pwd.txt`, select Read-Only, and then click OK.

   UNIX example:

   ```
   $ echo password > /tmp/pwd.txt
   ```

   ```
   $ chmod 400 /tmp/pwd.txt
   ```

To Install the JBoss Java Agent

1. Shut down the JBoss server where you plan to install the agent.
2. Make sure AM is running.
3. Run **agentadmin --install** to install the JBoss Java agent:

   ```
   $ /path/to/java_agents/jboss_agent/bin/agentadmin --install --acceptlicense
   ```

   a. When you run the command, you will be prompted to read and accept the software license agreement for the agent installation. You can suppress the license agreement prompt by including the `--acceptLicence` parameter. The inclusion of the option indicates that you have read and accepted the terms stated in the license. To view the license agreement, open `<server-root>/legal-notices/license.txt`.

   b. Enter the path to the JBoss installation directory. For example, `/path/to/jboss`.

   ```
   Enter the complete path to the home directory of the JBoss instance.
   [ ? : Help, ! : Exit ]
   Enter the path to the JBoss installation: /path/to/jboss
   ```

   c. Enter the JBoss deployment mode. Supported modes are `domain`, which allows you to manage multiple server instances from a single control point, or `standalone`, which is a single JBoss instance.

   ```
   Enter the name of the deployment mode of the JBoss installation that you wish
   to use with this agent. Supported values are: domain, standalone.
   [ ? : Help, < : Back, ! : Exit ]
   Enter the deployment mode of JBoss [standalone]: standalone
   ```

   If you chose `domain`, enter the name of the JBoss domain:

```
Enter the name of the profile to use in domain mode.
[ ? : Help, < : Back, ! : Exit ]
Enter the profile name: mydomain
```

d. Decide if you want to deploy the Java agent as a global JBoss module. If you want to include application-specific modules, enter `false`.

```
Enter true if you'd like to deploy the policy agent as a global JBoss module.
[ ? : Help, < : Back, ! : Exit ]
Install agent as global module? [true]:true
```

e. Enter the AM URL. For example, `https://openam.example.com:8443/openam`.

To balance agent connections to an AM site, configure the URL of the load balancer in front of the AM site.

Note

If your environment has a reverse proxy configured between AM and the agent, set the AM URL to the proxy URL instead. For example, `https://proxy.example.com:443/openam`. For more information about setting up the environment for reverse proxies, see "Configuring Apache HTTP Server as a Reverse Proxy Example".

```
Enter the URL where the AM server is running. Please include the
deployment URI also as shown below:
(http://openam.sample.com:58080/openam)
[ ? : Help, < : Back, ! : Exit ]
AM server URL: https://openam.example.com:8443/openam
```

f. Enter the Java agent URL. For example, `http://www.example.com:8080/agentapp`.

```
Enter the Agent URL. Please include the deployment URI also as shown below:
(http://agent1.sample.com:1234/agentapp)
[ ? : Help, < : Back, ! : Exit ]
Agent URL: http://www.example.com:8080/agentapp
```

g. Enter the agent profile name that you created in AM as part of the pre-installation procedure. For example, `JBossAgent`.

```
Enter the Agent profile name
[ ? : Help, < : Back, ! : Exit ]
Enter the Agent Profile name: JBossAgent
```

h. Enter the realm in which the specified agent profile exists.

Press **ENTER** to accept the default value of `/`, signifying the top-level realm.

Note

If you specify the Accept Empty value (^) option, the top-level realm is assumed.

```
Enter the Agent profile realm
[ ? : Help, < : Back, ! : Exit, ^ : Accept Empty value ]
Enter the Agent Profile realm [/]:
```

i. Enter the path to the password file that you created as part of the pre-installation procedure. For example, `/tmp/pwd.txt`.

```
Enter the path to a file that contains the password to be used for identifying
the Agent.
[ ? : Help, < : Back, ! : Exit ]
Enter the path to the password file: /tmp/pwd.txt
```

4. Review a summary of your responses and select an action to continue: install, go back a step, start over, or exit from the install:

```
-------------------------------------------
SUMMARY OF YOUR RESPONSES
-------------------------------------------
JBoss home directory : /path/to/jboss/
JBoss deployment mode: standalone
Install agent as global module: true
AM server URL : https://openam.example.com:8443/openam

Agent URL : http://www.example.com:8080/agentapp
Agent Profile name : JBossAgent
Agent Profile Realm : /
Agent Profile Password file name : /tmp/pwd.txt

Verify your settings above and decide from the choices below.
1. Continue with Installation
2. Back to the last interaction
3. Start Over
4. Exit
Please make your selection [1]: 1

...

SUMMARY OF AGENT INSTALLATION
-----------------------------
Agent instance name: Agent_001
Agent Bootstrap file location:
/path/to/java_agents/jboss_agent/Agent_001/config/
 OpenSSOAgentBootstrap.properties
Agent Configuration file location
/path/to/java_agents/jboss_agent/Agent_001/config/
 OpenSSOAgentConfiguration.properties
Agent Audit directory location:
/path/to/java_agents/jboss_agent/Agent_001/logs/audit
Agent Debug directory location:
/path/to/java_agents/jboss_agent/Agent_001/logs/debug
```

```
Install log file location:
/path/to/java_agents/jboss_agent/installer-logs/audit/install.log

Thank you for using AM Policy Agent
```

Upon successful completion, the installer updates the JBoss configuration, adds the Java agent application under `JBOSS_HOME/server/standalone/deployments`, and also sets up configuration and log directories for the Java agent.

5. Take note of the configuration files and log locations.

   Each Java agent instance that you install on the system has its own numbered configuration and logs directory. The first Java agent instance configuration files and logs are thus located under the directory `java_agents/jboss_agent/Agent_001/`:

   `config/OpenSSOAgentBootstrap.properties`

   Used to bootstrap the Java agent, allowing it to connect to AM and download its configuration.

   `config/OpenSSOAgentConfiguration.properties`

   Only used if you configured the Java agent to use local configuration.

   `logs/audit/`

   Operational audit log directory, only used if remote logging to AM is disabled.

   `logs/debug/`

   Debug directory where the debug file resides. Useful in troubleshooting Java agent issues.

6. To protect an application in the container, configure the agent filter.

7. (Optional) If you responded `false` to the `Deploy the policy agent as a global JBoss module` question during the installation process, perform the following steps:

   a. Add the following line to the */path/to/protected/app/*`META-INF/MANIFEST.MF` file of the application:

   ```
   Dependencies: org.forgerock.openam.agent
   ```

   b. Create a */path/to/protected/app/*`WEB-INF/jboss-deployment-structure.xml` file with the following content:

```
<?xml version="1.0"?>
 <jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
  <deployment>
   <dependencies>
    <module name="org.forgerock.openam.agent" >
     <imports>
      <include path="META-INF**"/>
      <include path="org**"/>
     </imports>
    </module>
   </dependencies>
  </deployment>
</jboss-deployment-structure>
```

8. (Optional) If you responded `domain` to the `Enter the name of the deployment mode` question during the installation process, you must manually deploy the `java_agents/jboss_agent/etc/agentapp.war` file to JBoss.

   The reason manual deployment is required when running JBoss in domain mode is that the agent installer uses auto-deployment capabilities provided by the JBoss deployment scanner. The deployment scanner is used only in standalone mode. When running JBoss in standalone mode, it is not necessary to manually deploy the `agentapp.war` file.

9. (Optional) If you have a policy configured, you can test your agent installation. For example, try to browse to a resource that your agent protects. You should be redirected to AM to authenticate, for example, as user `demo`, password `changeit`. After you authenticate, AM then redirects you back to the resource you tried to access.

Installing the JBoss Java Agent Silently

To install the JBoss Java agent silently, you must create a response file containing the installation parameters that you will then provide to the **agentadmin** command.

The following is an example of the response file to install the agent when JBoss is configured in `standalone` mode:

```
# Agent User Response File
HOME_DIR= /usr/share/JBoss_7
INSTANCE_NAME= standalone
GLOBAL_MODULE= true
INSTALL_PROFILE_NAME=
AM_SERVER_URL= https://openam.example.com:8443/openam
AGENT_URL= http://www.example.com:8080/agentapp
AGENT_PROFILE_NAME= JBossAgent
AGENT_PROFILE_REALM= /
AGENT_PASSWORD_FILE= /tmp/pwd.txt
```

The `INSTALL_PROFILE_NAME` variable is only used when the `INSTANCE_NAME` is set to `domain`, and it specifies the name of the JBoss domain profile.

To balance agent connections to an AM site, set the `AM_SERVER_URL` variable as the URL of the load balancer in front of the AM site.

Note

If your environment has a reverse proxy configured between AM and the agent, set the AM URL to the proxy URL instead. For example, `https://proxy.example.com:443/openam`. For more information about setting up the environment for reverse proxies, see "Configuring Apache HTTP Server as a Reverse Proxy Example".

You can also create this file automatically when installing the agent by running the **agentadmin** command with the `--saveResponse` option. For example:

```
$ agentadmin --install --saveResponse response-file
```

Complete the following procedures to install the JBoss Java agent silently:

To Complete Pre-Installation Tasks

Perform the following steps to create the configuration required by the Java agent before installing it:

1. Create at least one policy in AM to protect resources with the agent, as described in the procedure *Implementing Authorization Using the Access Management Console*.
2. Create an agent profile in AM, required by the Java agent to connect and communicate with AM. For more information, see "Creating Agent Profiles".
3. Ensure that the key pair configured for signing the OpenID Connect JWTs exchanged between AM and the Java agents is not the default `test` key pair. For more information, see "Configuring Access Management Servers to Communicate With Java Agents".
4. Configure AM to protect the cross-domain single sign-on (CDSSO) session cookie from hijacking. For more information, see Implementing Cross-Domain Single Sign-On in the *ForgeRock Access Management Authentication and Single Sign-On Guide*.
5. Consider the communication between the agents and the AM servers, and between the agents and the clients when installing agents in environments with load balancers and/or reverse proxies. For more information, see "*Configuring Environments With Load Balancers and Reverse Proxies*".
6. Create a text file containing only the password specified when creating the agent profile, and protect it:

   Windows example:

   ```
   C:\> echo password > pwd.txt
   ```

   In Windows Explorer, right-click the password file, for example `pwd.txt`, select Read-Only, and then click OK.

   UNIX example:

```
$ echo password > /tmp/pwd.txt
```

```
$ chmod 400 /tmp/pwd.txt
```

To Install the JBoss Java Agent Silently

1. Review the information in "Before You Install" before proceeding.
2. Shut down the JBoss server where you plan to install the agent.
3. Make sure AM is running.
4. Make sure you have a response file ready. For example, `response-file`. For more information, see "Installing the JBoss Java Agent Silently".
5. Run the **agentadmin** command with the `--useResponse` option. For example:

   ```
   $ agentadmin --install --acceptLicense --useResponse response-file
   ```

6. To protect an application in the container, configure the agent filter.
7. If you configured the `GLOBAL_MODULE` variable as `false` in the response file, add the following line to the `META-INF/MANIFEST.MF` file of the application:

   ```
   Dependencies: org.forgerock.openam.agent
   ```

8. If you configured the `INSTANCE_NAME` variable as `domain` in the response file, you must manually deploy the `java_agents/jboss_agent/etc/agentapp.war` file to JBoss.

   The reason manual deployment is required when running JBoss in domain mode is that the agent installer uses auto-deployment capabilities provided by the JBoss deployment scanner. The deployment scanner is used only in standalone mode. When running JBoss in standalone mode, it is not necessary to manually deploy the `agentapp.war` file.

Installing the Jetty Java Agent

f This section covers prerequisites and installation procedures for Java Agents 5.6 on Jetty.

Before You Install

1. Download the Java agent from BackStage. For more information, see "Downloading and Unzipping Java Agents".
2. Consider the following points before installing the Jetty Java agent:
   - Install Jetty before you install the agent.

- Install a supported version of the Java runtime environment, as described in "Java Requirements" in the *Release Notes*. Set the `JAVA_HOME` environment variable accordingly. The agent installer requires Java.

```
$ echo $JAVA_HOME
/path/to/java
```

- Command-line examples in this chapter show Jetty accessed remotely. If you are following the examples and have issues accessing Jetty remotely, you might have to change filter settings in the deployment descriptor file, such as `/path/to/jetty/webapps/test/WEB-INF/web.xml`, as shown in the following example:

```
 <filter>
 <filter-name>TestFilter</filter-name>
 <filter-class>com.acme.TestFilter</filter-class>
 <init-param>
   <param-name>remote</param-name>
   <param-value>true</param-value> <!-- default: false -->
 </init-param>
</filter>
```

Installing the Jetty Java Agent

Complete the following procedures to install the Jetty Java agent:

To Complete Pre-Installation Tasks

Perform the following steps to create the configuration required by the Java agent before installing it:

1. Create at least one policy in AM to protect resources with the agent, as described in the procedure *Implementing Authorization Using the Access Management Console*.
2. Create an agent profile in AM, required by the Java agent to connect and communicate with AM. For more information, see "Creating Agent Profiles".
3. Ensure that the key pair configured for signing the OpenID Connect JWTs exchanged between AM and the Java agents is not the default `test` key pair. For more information, see "Configuring Access Management Servers to Communicate With Java Agents".
4. Configure AM to protect the cross-domain single sign-on (CDSSO) session cookie from hijacking. For more information, see Implementing Cross-Domain Single Sign-On in the *ForgeRock Access Management Authentication and Single Sign-On Guide*.
5. Consider the communication between the agents and the AM servers, and between the agents and the clients when installing agents in environments with load balancers and/or reverse proxies. For more information, see "*Configuring Environments With Load Balancers and Reverse Proxies*".
6. Create a text file containing only the password specified when creating the agent profile, and protect it:

Windows example:

```
C:\> echo password > pwd.txt
```

In Windows Explorer, right-click the password file, for example `pwd.txt`, select Read-Only, and then click OK.

UNIX example:

```
$ echo password > /tmp/pwd.txt
```

```
$ chmod 400 /tmp/pwd.txt
```

To Install the Jetty Java Agent

1. Shut down the Jetty server where you plan to install the agent.
2. Make sure AM is running.
3. Run **agentadmin --install** to install the agent:

   ```
   $ /path/to/java_agents/jetty_agent/bin/agentadmin --install --acceptLicense
   ```

   a. When you run the command, you will be prompted to read and accept the software license agreement for the agent installation. You can suppress the license agreement prompt by including the `--acceptLicence` parameter. The inclusion of the option indicates that you have read and accepted the terms stated in the license. To view the license agreement, open `<server-root>/legal-notices/license.txt`.

   b. Enter the path to the Jetty configuration directory. For example, `/path/to/jetty/etc`.

   ```
   Enter the complete path to the directory which is used by Jetty Server to store
   its configuration Files. This directory uniquely identifies the Jetty
   Server instance that is secured by this Agent.
   [ ? : Help, ! : Exit ]
   Enter the Jetty Server Config Directory Path [/opt/jetty/etc]: /path/to/jetty/etc
   ```

   c. Enter the path to the Jetty root directory. For example, `/path/to/jetty`.

   ```
   This is the root of the Jetty installation.
   [ ? : Help, < : Back, ! : Exit ]
   Enter the Jetty home directory path: /path/to/jetty
   ```

   d. Enter the AM URL. For example, `https://openam.example.com:8443/openam`.

   To balance agent connections to an AM site, configure the URL of the load balancer in front of the AM site.

   Note

   If your environment has a reverse proxy configured between AM and the agent, set the AM URL to the proxy URL instead. For example, `https://proxy.example.com:443/openam`. For more information about

setting up the environment for reverse proxies, see "Configuring Apache HTTP Server as a Reverse Proxy Example".

```
Enter the URL where the AM server is running. Please include the
deployment URI also as shown below:
(http://openam.sample.com:58080/openam)
[ ? : Help, < : Back, ! : Exit ]
AM server URL: https://openam.example.com:8443/openam
```

e. Enter the agent URL. For example, `http://www.example.com:8080/agentapp`.

```
Enter the Agent URL. Please include the deployment URI also as shown below:
(http://agent1.sample.com:1234/agentapp)
[ ? : Help, < : Back, ! : Exit ]
Agent URL: http://www.example.com:8080/agentadmin
```

f. Enter the agent profile name that you created in AM as part of the pre-installation procedure. For example, `JettyAgent`.

```
Enter the Agent profile name
[ ? : Help, < : Back, ! : Exit ]
Enter the Agent Profile name: JettyAgent
```

g. Enter the realm in which the specified agent profile exists.

Press **ENTER** to accept the default value of `/`, signifying the top-level realm.

Note

If you specify the Accept Empty value (^) option, the top-level realm is assumed.

```
Enter the Agent profile realm
[ ? : Help, < : Back, ! : Exit, ^ : Accept Empty value ]
Enter the Agent Profile realm [/]:
```

h. Enter the path to the password file you created as part of the pre-installation procedure. For example, `/tmp/pwd.txt`.

```
Enter the path to a file that contains the password to be used for identifying
the Agent.
[ ? : Help, < : Back, ! : Exit ]
Enter the path to the password file: /tmp/pwd.txt
```

4. Review a summary of your responses and select an action to continue: install, go back a step, start over, or exit from the installer:

```
---------------------------------------------
SUMMARY OF YOUR RESPONSES
---------------------------------------------
Jetty Server Config Directory : /path/to/jetty/etc
AM server URL : https://openam.example.com:8443/openam
Jetty installation directory. : /path/to/jetty
```

```
Agent URL : https://www.example.com:8443/agentapp
Agent Profile name : JettyAgent
Agent Profile Realm : /
Agent Profile Password file name : /tmp/pwd.txt

Verify your settings above and decide from the choices below.
1. Continue with Installation
2. Back to the last interaction
3. Start Over
4. Exit
Please make your selection [1]: 1

...

SUMMARY OF AGENT INSTALLATION
-----------------------------
Agent instance name: Agent_001
Agent Bootstrap file location:
/path/to/java_agents/jetty_agent/Agent_001/config/
 OpenSSOAgentBootstrap.properties
Agent Configuration file location
/path/to/java_agents/jetty_agent/Agent_001/config/
 OpenSSOAgentConfiguration.properties
Agent Audit directory location:
/path/to/java_agents/jetty_agent/Agent_001/logs/audit
Agent Debug directory location:
/path/to/java_agents/jetty_agent/Agent_001/logs/debug


Install log file location:
/path/to/java_agents/jetty_agent/installer-logs/audit/install.log

 Thank you for using AM Policy Agent
```

Upon successful completion, the installer updates Jetty's `start.jar` to reference the agent, sets up the agent web application, and also sets up configuration and log directories for the agent.

5. Take note of the configuration files and log locations.

   Each agent instance that you install on the system has its own numbered configuration and logs directory. The first agent's configuration and logs are thus located under the directory `java_agents/jetty_agent/Agent_001/`:

   `config/OpenSSOAgentBootstrap.properties`

   Used to bootstrap the Java agent, allowing the agent to connect to AM and download its configuration.

   `config/OpenSSOAgentConfiguration.properties`

   Only used if you configured the Java agent to use local configuration.

   `logs/audit/`

   Operational audit log directory, only used if remote logging to AM is disabled.

`logs/debug/`

Debug directory where the `debug.out` debug file resides. Useful in troubleshooting Java agent issues.

6. To protect an application in the container, configure the agent filter.

7. (Optional) If you have a policy configured, you can test the agent installation. For example, try to browse to a resource that your agent protects. You should be redirected to AM to authenticate, for example, as user `demo`, password `changeit`. After you authenticate, AM then redirects you back to the resource you tried to access.

Installing the Jetty Java Agent Silently

To install the Jetty Java agent silently, you must create a response file contanining the installation parameters and then provide it to the **agentadmin** command.

The following is an example of the response file:

```
# Agent User Response File
CONFIG_DIR= /usr/local/jetty-distribution-9.4.7/etc
JETTY_HOME= /usr/local/jetty-distribution-9.4.7
AM_SERVER_URL= https://openam.example.com:8443/openam
AGENT_URL= http://www.example.com:8080/agentapp
AGENT_PROFILE_NAME= JettyAgent
AGENT_PROFILE_REALM= /
AGENT_PASSWORD_FILE= /tmp/pwd.txt
```

To balance agent connections to an AM site, set the `AM_SERVER_URL` variable as the URL of the load balancer in front of the AM site.

Note

If your environment has a reverse proxy configured between AM and the agent, set the AM URL to the proxy URL instead. For example, `https://proxy.example.com:443/openam`. For more information about setting up the environment for reverse proxies, see "Configuring Apache HTTP Server as a Reverse Proxy Example".

You can also create this file automatically when installing the Java agent by running the **agentadmin** command with the `--saveResponse` option. For example:

```
$ agentadmin --install --saveResponse response-file
```

Complete the following procedures to install the Jetty Java agent silently:

To Complete Pre-Installation Tasks

Perform the following steps to create the configuration required by the Java agent before installing it:

1. Create at least one policy in AM to protect resources with the agent, as described in the procedure *Implementing Authorization Using the Access Management Console*.
2. Create an agent profile in AM, required by the Java agent to connect and communicate with AM. For more information, see "Creating Agent Profiles".
3. Ensure that the key pair configured for signing the OpenID Connect JWTs exchanged between AM and the Java agents is not the default `test` key pair. For more information, see "Configuring Access Management Servers to Communicate With Java Agents".
4. Configure AM to protect the cross-domain single sign-on (CDSSO) session cookie from hijacking. For more information, see Implementing Cross-Domain Single Sign-On in the *ForgeRock Access Management Authentication and Single Sign-On Guide*.
5. Consider the communication between the agents and the AM servers, and between the agents and the clients when installing agents in environments with load balancers and/or reverse proxies. For more information, see "*Configuring Environments With Load Balancers and Reverse Proxies*".
6. Create a text file containing only the password specified when creating the agent profile, and protect it:

   Windows example:

   ```
   C:\> echo password > pwd.txt
   ```

   In Windows Explorer, right-click the password file, for example `pwd.txt`, select Read-Only, and then click OK.

   UNIX example:

   ```
   $ echo password > /tmp/pwd.txt
   ```

   ```
   $ chmod 400 /tmp/pwd.txt
   ```

To install the Jetty Java Agent Silently

1. Check the information in "Before You Install".
2. Shut down the Jetty server where you plan to install the agent.
3. Make sure that AM is running.
4. Make sure you have a response file ready. For example, `response-file`. For more information, see "Installing the Jetty Java Agent Silently".
5. Run the **agentadmin** command with the `--useResponse` option. For example:

   ```
   $ agentadmin --install --acceptLicense --useResponse response-file
   ```

6. To protect an application in the container, configure the agent filter.

Installing the WebLogic Java Agent

This section covers prerequisites and installation procedures for Java Agents 5.6 on WebLogic.

Before You Install

1. Download the agent from BackStage. For more information, see "Downloading and Unzipping Java Agents".
2. Consider the following points before installing the WebLogic Java agent:
   - Install WebLogic before you install the agent.
   - Install a supported version of the Java runtime environment, as described in "Java Requirements" in the *Release Notes*. Set the `JAVA_HOME` environment variable accordingly. The agent installer requires Java.

     ```
     $ echo $JAVA_HOME
     /path/to/java
     ```

Installing the WebLogic Java Agent

Complete the following procedures to install the WebLogic Java agent:

To Complete Pre-Installation Tasks

Perform the following steps to create the configuration required by the Java agent before installing it:

1. Create at least one policy in AM to protect resources with the agent, as described in the procedure *Implementing Authorization Using the Access Management Console*.
2. Create an agent profile in AM, required by the Java agent to connect and communicate with AM. For more information, see "Creating Agent Profiles".
3. Ensure that the key pair configured for signing the OpenID Connect JWTs exchanged between AM and the Java agents is not the default `test` key pair. For more information, see "Configuring Access Management Servers to Communicate With Java Agents".
4. Configure AM to protect the cross-domain single sign-on (CDSSO) session cookie from hijacking. For more information, see Implementing Cross-Domain Single Sign-On in the *ForgeRock Access Management Authentication and Single Sign-On Guide*.
5. Consider the communication between the agents and the AM servers, and between the agents and the clients when installing agents in environments with load balancers and/or reverse proxies. For more information, see "*Configuring Environments With Load Balancers and Reverse Proxies*".
6. Create a text file containing only the password specified when creating the agent profile, and protect it:

Windows example:

```
C:\> echo password > pwd.txt
```

In Windows Explorer, right-click the password file, for example `pwd.txt`, select Read-Only, and then click OK.

UNIX example:

```
$ echo password > /tmp/pwd.txt
```

```
$ chmod 400 /tmp/pwd.txt
```

To Install the WebLogic Java Agent

1. Shut down the WebLogic server where you plan to install the agent.
2. Make sure AM is running.
3. Run **agentadmin --install** to install the agent:

   ```
   $ /path/to/java_agents/weblogic_agent/bin/agentadmin --install --acceptlicense
   ```

   a. When you run the command, you will be prompted to read and accept the software license agreement for the agent installation. You can suppress the license agreement prompt by including the `--acceptLicence` parameter. The inclusion of the option indicates that you have read and accepted the terms stated in the license. To view the license agreement, open `<server-root>/legal-notices/license.txt`.

   b. Enter the path to the `startWebLogic.sh` file of the WebLogic domain where you want to install the agent. For example, `/Oracle_Home/user_projects/domains/base_domain/startWebLogic.sh`.

   ```
   Enter the path to the location of the script used to start the WebLogic domain.
   Please ensure that the agent is first installed on the admin server instance
   before installing on any managed server instance.
   [ ? : Help, ! : Exit ]
   Enter the Startup script location
   [/usr/local/bea/user_projects/domains/base_domain/startWebLogic.sh]: /Oracle_Home/user_projects/
   domains/base_domain/startWebLogic.sh
   ```

   c. Enter the path to the WebLogic installation directory. For example, `/path/to/weblogic`.

   ```
   Enter the WebLogic home directory
   [ ? : Help, < : Back, ! : Exit ]
   Enter the WebLogic home directory [/usr/local/bea/wlserver_10.0]: /path/to/weblogic
   ```

   d. Enter the AM URL. For example, `https://openam.example.com:8443/openam`.

   To balance agent connections to an AM site, configure the URL of the load balancer in front of the AM site.

Note

If your environment has a reverse proxy configured between AM and the agent, set the AM URL to the proxy URL instead. For example, `https://proxy.example.com:443/openam`. For more information about setting up the environment for reverse proxies, see "Configuring Apache HTTP Server as a Reverse Proxy Example".

```
Enter the URL where the AM server is running. Please include the
deployment URI also as shown below:
(http://openam.sample.com:58080/openam)
[ ? : Help, < : Back, ! : Exit ]
AM server URL: https://openam.example.com:8443/openam
```

To balance agent connections to an AM site, configure the URL of the load balancer in front of the AM site.

e. Enter the agent URL. For example, `http://www.example.com:8080/agentapp`.

```
Enter the Agent URL. Please include the deployment URI also as shown below:
(http://agent1.sample.com:1234/agentapp)
[ ? : Help, < : Back, ! : Exit ]
Agent URL: http://www.example.com:8080/agentapp
```

f. Enter the agent profile name that you created in AM as part of the pre-installation procedure. For example, `WebLogicAgent`.

```
Enter the Agent profile name
[ ? : Help, < : Back, ! : Exit ]
Enter the Agent Profile name: WebLogicAgent
```

g. Enter the realm in which the specified agent profile exists.

Press **ENTER** to accept the default value of `/`, signifying the top-level realm.

Note

If you specify the Accept Empty value (^) option, the top-level realm is assumed.

```
Enter the Agent profile realm
[ ? : Help, < : Back, ! : Exit, ^ : Accept Empty value ]
Enter the Agent Profile realm [/]:
```

h. Enter the path to the password file that you created as part of the pre-installation procedure. For example, `/tmp/pwd.txt`.

```
Enter the path to a file that contains the password to be used for identifying
the Agent.
[ ? : Help, < : Back, ! : Exit ]
Enter the path to the password file: /tmp/pwd.txt
```

4. Review a summary of your responses and select an action to continue: install, go back a step, start over, or exit from the install:

```
$ /path/to/java_agents/weblogic_agent/bin/agentadmin --install --acceptLicense
------------------------------------------------
SUMMARY OF YOUR RESPONSES
------------------------------------------------
Startup script location :
/Oracle_Home/user_projects/domains/base_domain/startWebLogic.sh
WebLogic Server instance name : AdminServer
WebLogic home directory : /path/to/weblogic
AM server URL : https://openam.example.com:8443/openam

Agent URL : http://www.example.com:8080/agentapp
Agent Profile name : WebLogicAgent
Agent Profile Realm : /
Agent Profile Password file name : /tmp/pwd.txt

Verify your settings above and decide from the choices below.
1. Continue with Installation
2. Back to the last interaction
3. Start Over
4. Exit
Please make your selection [1]: 1

...

SUMMARY OF AGENT INSTALLATION
-----------------------------
Agent instance name: Agent_001
Agent Bootstrap file location:
/path/to/java_agents/weblogic_agent/Agent_001/config/
 OpenSSOAgentBootstrap.properties
Agent Configuration file location
/path/to/java_agents/weblogic_agent/Agent_001/config/
 OpenSSOAgentConfiguration.properties
Agent Audit directory location:
/path/to/java_agents/weblogic_agent/Agent_001/logs/audit
Agent Debug directory location:
/path/to/java_agents/weblogic_agent/Agent_001/logs/debug


Install log file location:
/path/to/java_agents/weblogic_agent/installer-logs/audit/install.log

Thank you for using AM Policy Agent
```

5. Take note of the configuration files and log locations.

   Each agent instance that you install on the system has its own numbered configuration and logs directory. The first agent's configuration and logs are thus located under the directory `java_agents/weblogic_agent/Agent_001/`:

`config/OpenSSOAgentBootstrap.properties`

Used to bootstrap the Java agent, allowing the agent to connect to AM and download its configuration.

`config/OpenSSOAgentConfiguration.properties`

Only used if you configured the Java agent to use local configuration.

`logs/audit/`

Operational audit log directory, only used if remote logging to AM is disabled.

`logs/debug/`

Debug directory where the debug file resides. Useful in troubleshooting agent issues.

6. The agent requires sourcing before it will work properly. There are two ways to source:

   - Manually source the file containing the agent environment settings for WebLogic before starting the application server.

     ```
     $ . /path/to/setAgentEnv_AdminServer.sh
     ```

   - Or edit the `startWebLogic.sh` script to set the sourcing needed for the agent, by adding these lines after the code block shown. Add the setAgentEnv_AdminServer.sh line to the following location in the file. The drawback to this approach is that it could be overwritten, as noted in the file:

     ```
     $ cat /path/to/startWebLogic.sh
     ...
     # Any changes to this script may be lost when adding extensions to this
     # configuration.
     DOMAIN_HOME="/opt/Oracle/Middleware/user_projects/domains/base_domain"
      . /path/to/setAgentEnv_AdminServer.sh
     ${DOMAIN_HOME}/bin/startWebLogic.sh $*
     ```

   Note

   If the sourcing is not set properly, the following message appears:

   ```
   <Error> <HTTP> <cent.example.com>
   <AdminServer> <[STANDBY] ExecuteThread: '5' for queue: 'weblogic.kernel.
   Default (self-tuning)'> <<WLS Kernel>> <><> <> <1360800613441>
   <BEA-101165> <Could not load user defined filter in web.xml:
   ServletContext@1761850405[app:agentapp module:agentapp.war path:null
   spec-version:null] com.sun.identity.agents.filter.AmAgentFilter.
   java.lang.ClassNotFoundException:
   com.sun.identity.agents.filter.AmAgentFilter
   ```

7. Start the WebLogic server.

8. Deploy the `/path/to/java_agents/weblogic_agent/etc/agentapp.war` agent application in WebLogic.
9. To protect an application in the container, configure the agent filter.
10. (Optional) If you have a policy configured, you can test your agent installation. For example, try to browse to a resource that your agent protects. You should be redirected to AM to authenticate, for example, as user `demo`, password `changeit`. After you authenticate, AM then redirects you back to the resource you tried to access.

Installing the WebLogic Java Agent Silently

To install the WebLogic Java agent silently, you must create a response file containing the installation parameters that you will then provide to the **agentadmin** command.

The following is an example of the response file:

```
# Agent User Response File
STARTUP_SCRIPT= /Oracle_Home/user_projects/domains/base_domain/startWebLogic.sh
SERVER_NAME= AdminServer
WEBLOGIC_HOME_DIR= /usr/local/weblogic12
AM_SERVER_URL= https://openam.example.com:8443/openam
AGENT_URL= http://www.example.com:8080/agentapp
AGENT_PROFILE_NAME= WebLogicAgent
AGENT_PROFILE_REALM= /
AGENT_PASSWORD_FILE= /tmp/pwd.txt
```

To balance agent connections to an AM site, set the `AM_SERVER_URL` variable as the URL of the load balancer in front of the AM site.

Note

If your environment has a reverse proxy configured between AM and the agent, set the AM URL to the proxy URL instead. For example, `https://proxy.example.com:443/openam`. For more information about setting up the environment for reverse proxies, see "Configuring Apache HTTP Server as a Reverse Proxy Example".

You can also create this file automatically when installing the Java agent by running the **agentadmin** command with the `--saveResponse` option. For example:

```
$ agentadmin --install --saveResponse response-file
```

Complete the following procedures to install the WebLogic Java agent silently:

To Complete Pre-Installation Tasks

Perform the following steps to create the configuration required by the Java agent before installing it:

1. Create at least one policy in AM to protect resources with the agent, as described in the procedure *Implementing Authorization Using the Access Management Console*.
2. Create an agent profile in AM, required by the Java agent to connect and communicate with AM. For more information, see "Creating Agent Profiles".

3. Ensure that the key pair configured for signing the OpenID Connect JWTs exchanged between AM and the Java agents is not the default `test` key pair. For more information, see "Configuring Access Management Servers to Communicate With Java Agents".

4. Configure AM to protect the cross-domain single sign-on (CDSSO) session cookie from hijacking. For more information, see Implementing Cross-Domain Single Sign-On in the *ForgeRock Access Management Authentication and Single Sign-On Guide*.

5. Consider the communication between the agents and the AM servers, and between the agents and the clients when installing agents in environments with load balancers and/or reverse proxies. For more information, see "*Configuring Environments With Load Balancers and Reverse Proxies*".

6. Create a text file containing only the password specified when creating the agent profile, and protect it:

   Windows example:

   ```
   C:\> echo password > pwd.txt
   ```

   In Windows Explorer, right-click the password file, for example `pwd.txt`, select Read-Only, and then click OK.

   UNIX example:

   ```
   $ echo password > /tmp/pwd.txt
   ```

   ```
   $ chmod 400 /tmp/pwd.txt
   ```

To Install the WebLogic Java Agent Silently

1. Review the information in "Before You Install".

2. Shut down the WebLogic server where you plan to install the agent.

3. Make sure AM is running.

4. Run the **agentadmin** command with the `--useResponse` option. For example:

   ```
   $ agentadmin --install --acceptLicense --useResponse response-file
   ```

5. The agent requires sourcing before it will work properly. There are two ways to source:

   - Manually source the file containing the agent environment settings for WebLogic before starting the application server.

     ```
     $ . /path/to/setAgentEnv_AdminServer.sh
     ```

   - Or edit the `startWebLogic.sh` script to set the sourcing needed for the agent, by adding these lines after the code block shown. Add the setAgentEnv_AdminServer.sh line to the following

location in the file. The drawback to this approach is that it could be overwritten, as noted in the file:

```
$ cat /path/to/startWebLogic.sh
...
# Any changes to this script may be lost when adding extensions to this
# configuration.
DOMAIN_HOME="/opt/Oracle/Middleware/user_projects/domains/base_domain"
. /path/to/setAgentEnv_AdminServer.sh
${DOMAIN_HOME}/bin/startWebLogic.sh $*
```

6. Start the WebLogic Server.
7. Deploy the `/path/to/java_agents/weblogic_agent/etc/agentapp.war` agent application in WebLogic.
8. To protect an application in the container, configure the agent filter.

Installing the WebLogic Java Agent in Multi-Server Domains

In many WebLogic domains, the administration server provides a central point for controlling and managing the configuration of the managed servers that host protected applications.

If WebLogic-managed servers run on different hosts, you must create separate agent profiles and perform separate installations for each so that AM can send notifications to the appropriate addresses.

To Install the WebLogic Java Agent on Administration and Managed Servers

For multi-server WebLogic domains, install the Java agent as follows:

1. If servers are on different hosts, create agent profiles for each server where you plan to install the agent.

   The steps are described under "Installing the WebLogic Java Agent".
2. Prepare your protected web applications by adding the agent filter configuration as described in "Configuring the Agent Filter for an Application".
3. Use the **agentadmin** command to install the agent either interactively, or silently on each server in the domain:

   - For interactive installation, follow the instructions in "To Install the WebLogic Java Agent".
   - For silent installation, follow the instructions in "Installing the WebLogic Java Agent Silently".
4. On each managed server in the domain, update the classpath to include agent .jar files.

   In WebLogic Node Manager console, navigate to Environment > Servers > *server* > Server Start > Class Path, and then edit the classpath as in the following example, but all on a single line:

```
/path/to/java_agents/weblogic_agent/lib/agent.jar:
/path/to/java_agents/weblogic_agent/lib/openssoclientsdk.jar:
 ...
/path/to/java_agents/weblogic_agent/locale:
/path/to/java_agents/weblogic_agent/Agent_001/config:
$CLASSPATH
```

Replace the paths in the example with the actual paths for your domain.

5. Restart the managed servers.

Installing the WebSphere Java Agent

This section covers prerequisites and installation procedures for Java Agents 5.6 on WebSphere.

Before You Install

1. Download the agent from BackStage. For more information, see "Downloading and Unzipping Java Agents".
2. Consider the following points before installing the WebSphere Java agent:
   - Install WebSphere before you install the agent.
   - Install a supported version of the Java runtime environment, as described in "Java Requirements" in the *Release Notes*. Set the `JAVA_HOME` environment variable accordingly. The agent installer requires Java.

     ```
     $ echo $JAVA_HOME
     /path/to/java
     ```

   - If you are using IBM Java, perform the following procedure:

     ### *To Install With IBM Java*

     The WebSphere Java agent runs with IBM Java. To install the agent using IBM Java on platforms other than AIX, you must change the **agentadmin** script to use the IBM Java Cryptography Extensions (JCE).

     Note that line breaks and continuation marker (\) characters have been manually added to the following examples to aid display in the documentation. These are not required when editing the script.

     1. Open the file `bin/agentadmin` for editing.
     2. Edit the line that calls the **AdminToolLauncher** jar file to move the `$AGENT_OPTS` environment variable before the classpath is set:

Before:

```
$JAVA_VM -classpath "$AGENT_CLASSPATH" $AGENT_OPTS \
        com.sun.identity.install.tools.launch.AdminToolLauncher $*
```

After:

```
$JAVA_VM $AGENT_OPTS -classpath "$AGENT_CLASSPATH" \
        com.sun.identity.install.tools.launch.AdminToolLauncher $*
```

3. Save your work.

   You can now install the WebSphere Java agent with IBM Java as described in "Installing the WebSphere Java Agent".

Installing the WebSphere Java Agent

Complete the following procedures to install the WebSphere Java agent:

To Complete Pre-Installation Tasks

Perform the following steps to create the configuration required by the Java agent before installing it:

1. Create at least one policy in AM to protect resources with the agent, as described in the procedure *Implementing Authorization Using the Access Management Console*.

2. Create an agent profile in AM, required by the Java agent to connect and communicate with AM. For more information, see "Creating Agent Profiles".

3. Ensure that the key pair configured for signing the OpenID Connect JWTs exchanged between AM and the Java agents is not the default `test` key pair. For more information, see "Configuring Access Management Servers to Communicate With Java Agents".

4. Configure AM to protect the cross-domain single sign-on (CDSSO) session cookie from hijacking. For more information, see Implementing Cross-Domain Single Sign-On in the *ForgeRock Access Management Authentication and Single Sign-On Guide*.

5. Consider the communication between the agents and the AM servers, and between the agents and the clients when installing agents in environments with load balancers and/or reverse proxies. For more information, see "*Configuring Environments With Load Balancers and Reverse Proxies*".

6. Create a text file containing only the password specified when creating the agent profile, and protect it:

   Windows example:

   ```
   C:\> echo password > pwd.txt
   ```

In Windows Explorer, right-click the password file, for example `pwd.txt`, select Read-Only, and then click OK.

UNIX example:

```
$ echo password > /tmp/pwd.txt
```

```
$ chmod 400 /tmp/pwd.txt
```

To Install the WebSphere Java Agent

1. Shut down the WebSphere server where you plan to install the agent.
2. Make sure AM is running.
3. Run **agentadmin --install** to install the agent:

```
$ /path/to/java_agents/websphere_agent/bin/agentadmin --install --acceptlicense
```

   a. When you run the command, you will be prompted to read and accept the software license agreement for the agent installation. You can suppress the license agreement prompt by including the `--acceptLicence` parameter. The inclusion of the option indicates that you have read and accepted the terms stated in the license. To view the license agreement, open `<server-root>/legal-notices/license.txt`.

   b. Enter the path to the configuration directory of the server instance for the WebSphere node. For example, `/path/to/WebSphere/AppServer/profiles/AppServ01/config/cells/DefaultCell01/nodes/DefaultNode01/servers/server1`.

```
Enter the fully qualified path to the configuration directory of the Server
Instance for the WebSphere node.
[ ? : Help, ! : Exit ]
Enter the Instance Config Directory
[/opt/IBM/WebSphere/AppServer/profiles/AppSrv01/config/cells/<hostname>Node01Cell/nodes/
<hostname>Node01/servers/server1]: /path/to/WebSphere/AppServer/profiles/AppServ01/config/cells/
DefaultCell01/nodes/DefaultNode01/servers/server1
```

   c. Enter the name of the server instance where the agent will be installed. For example, `server1`.

```
Enter the Server Instance name.
[ ? : Help, < : Back, ! : Exit ]
Enter the Server Instance name [server1]: server1
```

   d. Enter the path to the WebSphere install directory. For example, `/path/to/WebSphere/AppServer`.

```
Enter the WebSphere Install Root directory.
[ ? : Help, < : Back, ! : Exit ]
Enter the WebSphere Install Root directory
[/opt/IBM/WebSphere/AppServer]: /path/to/WebSphere/AppServer
```

   e. Enter the AM URL. For example, `https://openam.example.com:8443/openam`.

To balance agent connections to an AM site, configure the URL of the load balancer in front of the AM site.

Note

If your environment has a reverse proxy configured between AM and the agent, set the AM URL to the proxy URL instead. For example, `https://proxy.example.com:443/openam`. For more information about setting up the environment for reverse proxies, see "Configuring Apache HTTP Server as a Reverse Proxy Example".

```
Enter the URL where the AM server is running. Please include the
deployment URI also as shown below:
(http://openam.sample.com:58080/openam)
[ ? : Help, < : Back, ! : Exit ]
AM server URL: https://openam.example.com:8443/openam
```

f. Enter the agent URL. For example, `http://www.example.com:8080/agentapp`.

```
Enter the Agent URL. Please include the deployment URI also as shown below:
(http://agent1.sample.com:1234/agentapp)
[ ? : Help, < : Back, ! : Exit ]
Agent URL: http://www.example.com:8080/agentapp
```

g. Enter the agent profile name that you created in AM as part of the pre-installation procedure. For example, `WebSphereAgent`.

```
Enter the Agent profile name
[ ? : Help, < : Back, ! : Exit ]
Enter the Agent Profile name: WebSphereAgent
```

h. Enter the realm in which the specified agent profile exists.

Press **ENTER** to accept the default value of `/`, signifying the top-level realm.

Note

If you specify the Accept Empty value (`^`) option, the top-level realm is assumed.

```
Enter the Agent profile realm
[ ? : Help, < : Back, ! : Exit, ^ : Accept Empty value ]
Enter the Agent Profile realm [/]:
```

i. Enter the path to the password file that you created as part of the pre-installation procedure. For example, `/tmp/pwd.txt`.

```
Enter the path to a file that contains the password to be used for identifying
the Agent.
[ ? : Help, < : Back, ! : Exit ]
Enter the path to the password file: /tmp/pwd.txt
```

4. Review a summary of your responses and select an action to continue: install, go back a step, start over, or exit:

```
------------------------------------------------
SUMMARY OF YOUR RESPONSES
------------------------------------------------
Instance Config Directory :
/path/to/WebSphere/AppServer/profiles/AppServ01/config/cells/DefaultCell01/nodes/DefaultNode01/
servers/server1

Instance Server name : server1
WebSphere Install Root Directory : /path/to/WebSphere/AppServer
AM server URL : https://openam.example.com:8443/openam

Agent URL : http://www.example.com:8080/agentapp
Agent Profile name : WebSphereAgent
Agent Profile Realm : /
Agent Profile Password file name : /tmp/pwd.txt

Verify your settings above and decide from the choices below.
1. Continue with Installation
2. Back to the last interaction
3. Start Over
4. Exit
Please make your selection [1]: 1

...

SUMMARY OF AGENT INSTALLATION
-----------------------------
Agent instance name: Agent_001
Agent Bootstrap file location:
/path/to/java_agents/websphere_agent/Agent_001/config/
 OpenSSOAgentBootstrap.properties
Agent Configuration file location
/path/to/java_agents/websphere_agent/Agent_001/config/
 OpenSSOAgentConfiguration.properties
Agent Audit directory location:
/path/to/java_agents/websphere_agent/Agent_001/logs/audit
Agent Debug directory location:
/path/to/java_agents/websphere_agent/Agent_001/logs/debug


Install log file location:
/path/to/java_agents/websphere_agent/installer-logs/audit/install.log

Thank you for using AM Policy Agent
```

   Upon successful completion, the installer updates the WebSphere configuration, copies the agent libraries to WebSphere's external library directory, and also sets up configuration and log directories for the agent.

5. Take note of the configuration files and log locations.

Each agent instance that you install on the system has its own numbered configuration and logs directory. The first agent's configuration and logs are thus located under the directory `java_agents/websphere_agent/Agent_001/`:

`config/OpenSSOAgentBootstrap.properties`

Used to bootstrap the Java agent, allowing the agent to connect to AM and download its configuration.

`config/OpenSSOAgentConfiguration.properties`

Only used if you configured the Java agent to use local configuration.

`logs/audit/`

Operational audit log directory, only used if remote logging to AM is disabled.

`logs/debug/`

Debug directory where the debug file resides. Useful in troubleshooting agent issues.

6. Restart the WebSphere server.
7. Deploy the `/path/to/java_agents/websphere_agent/etc/agentapp.war` agent application in WebSphere.
8. To protect an application in the container, configure the agent filter.
9. (Optional) If you have a policy configured, you can test your agent installation. For example, try to browse to a resource that your agent protects. You should be redirected to AM to authenticate, for example, as user `demo`, password `changeit`. After you authenticate, AM then redirects you back to the resource you tried to access.

Installing the WebSphere Java Agent Silently

To install the WebSphere Java agent silently, you must create a response file containing the installation parameters that you will then provide to the **agentadmin** command.

The following is an example of the response file:

```
# Agent User Response File
SERVER_INSTANCE_DIR= /opt/IBM/WebSphere/AppServer/profiles/AppSrv01/config/cells/DefaultCell01/nodes/
DefaultNode01/servers/server1
SERVER_INSTANCE_NAME= server1
HOME_DIRECTORY= /opt/IBM/WebSphere/AppServer
AM_SERVER_URL= https://openam.example.com:8443/openam
AGENT_URL= http://www.example.com:8080/agentapp
AGENT_PROFILE_NAME= WebSphereAgent
AGENT_PROFILE_REALM= /
AGENT_PASSWORD_FILE= /tmp/pwd.txt
```

To balance agent connections to an AM site, set the `AM_SERVER_URL` variable as the URL of the load balancer in front of the AM site.

Note

If your environment has a reverse proxy configured between AM and the agent, set the AM URL to the proxy URL instead. For example, `https://proxy.example.com:443/openam`. For more information about setting up the environment for reverse proxies, see "Configuring Apache HTTP Server as a Reverse Proxy Example".

You can also create this file automatically when installing the Java agent by running the **agentadmin** command with the `--saveResponse` option. For example:

```
$ agentadmin --install --saveResponse response-file
```

Complete the following procedure to install the WebSphere Java agent silently:

To Complete Pre-Installation Tasks

Perform the following steps to create the configuration required by the Java agent before installing it:

1. Create at least one policy in AM to protect resources with the agent, as described in the procedure *Implementing Authorization Using the Access Management Console*.
2. Create an agent profile in AM, required by the Java agent to connect and communicate with AM. For more information, see "Creating Agent Profiles".
3. Ensure that the key pair configured for signing the OpenID Connect JWTs exchanged between AM and the Java agents is not the default `test` key pair. For more information, see "Configuring Access Management Servers to Communicate With Java Agents".
4. Configure AM to protect the cross-domain single sign-on (CDSSO) session cookie from hijacking. For more information, see Implementing Cross-Domain Single Sign-On in the *ForgeRock Access Management Authentication and Single Sign-On Guide*.
5. Consider the communication between the agents and the AM servers, and between the agents and the clients when installing agents in environments with load balancers and/or reverse proxies. For more information, see "*Configuring Environments With Load Balancers and Reverse Proxies*".
6. Create a text file containing only the password specified when creating the agent profile, and protect it:

   Windows example:

   ```
   C:\> echo password > pwd.txt
   ```

   In Windows Explorer, right-click the password file, for example `pwd.txt`, select Read-Only, and then click OK.

   UNIX example:

```
$ echo password > /tmp/pwd.txt
```

```
$ chmod 400 /tmp/pwd.txt
```

To Install the WebSphere Java Agent Silently

1. Check the information in "Before You Install".
2. Shut down the WebSphere server where you plan to install the agent.
3. Make sure AM is running.
4. Run the **agentadmin** command with the `--useResponse` option. For example:

   ```
   $ agentadmin --install --acceptLicense --useResponse response-file
   ```

5. Start the WebSphere server.
6. Deploy the `/path/to/java_agents/websphere_agent/etc/agentapp.war` agent application in WebSphere.
7. To protect an application in the container, configure the agent filter.
8. (Optional) If you have a policy configured, you can test your agent installation. For example, try to browse to a resource that your agent protects. You should be redirected to AM to authenticate, for example, as user `demo`, password `changeit`. After you authenticate, AM then redirects you back to the resource you tried to access.

Notes About WebSphere Network Deployment

When using WebSphere Application Server Network Deployment, you must install WebSphere Java agents on the Deployment Manager, on each Node Agent, and on each Application Server. Installation requires that you stop and then restart the Deployment Manager, each Node Agent, and each Application Server in the Network Deployment.

Before installation, synchronize each server configuration with the profile saved by the Deployment Manager using the **syncNode** command. After agent installation, copy the server configuration for each node stored in `server.xml` to the corresponding Deployment Manager profile. After you have synchronized the configurations, you must restart the Deployment Manager for the Network Deployment.

Chapter 5

Post-Installation Tasks

This chapter covers tasks to perform after installing Java agents in your environment. The following table contains a list of the tasks:

Task	Section
Configure the agent filter and mode of operation. You must configure the agent filter to protect your applications	Section
Configure Java agents to log audit messages	Section
Configure Java agents to provide performance monitoring metrics.	Section
Configure Java agents to communicate with AM using HTTPS	Section
Configure your environment when communication between clients and agents happens behind load balancers or reverse proxies	Section

Configuring the Agent Filter

The agent filter is a servlet that intercepts inbound client requests to a resource and processes them based on the filter mode of operation.

Configuring the agent filter is a two-step process:

- Configuring the Agent Filter for an Application
- Configuring the Agent Filter's Mode of Operation

Configuring the Agent Filter for an Application

The agent filter is configured in the application's `web.xml` file. Therefore, to protect several applications in the same container, you must configure the agent filter in each application.

Consider the following example configuration:

```
<filter>
  <filter-name>Agent</filter-name>
  <display-name>AM Agent</display-name>
  <description>AM Agent Filter</description>
  <filter-class>com.sun.identity.agents.filter.AmAgentFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>Agent</filter-name>
  <url-pattern>/*</url-pattern>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>INCLUDE</dispatcher>
  <dispatcher>FORWARD</dispatcher>
  <dispatcher>ERROR</dispatcher>
</filter-mapping>
```

The agent filter's configuration requires two elements:

- `filter`. Defines the unique identifier of the filter and the filter class. It contains the following elements:
  - `filter-name`. The value is a string, for example, `Agent`.
  - `display-name`. The value is a string, for example, `AM Agent`. The container's management console may use this string as an identifier for the filter.
  - `description`. The value is a string, for example, `AM Agent Filter`. The container's management console may use this string as description for the filter.
  - `filter-class`. The value is the agent filter class, `com.sun.identity.agents.filter.AmAgentFilter`.
- `filter-mapping`. Defines the resources protected by the filter. It contains the following elements:
  - `filter-name`. The value must match the value of the `filter-name` element defined in the `filter` element.
  - `url-pattern`. The value defines the resources that the agent protects. For example, set the value to `/*` to protect every resource in the application.
  - `dispatcher`. Optional. Set one or more `dispatcher` elements to protect the Java container dispatchers as well as the application.

    Refer to the container vendor's documentation for more information about the container's dispatchers.

If you configure additional filters in the `web.xml` file, ensure the agent filter is defined first.

Configuring the Agent Filter's Modes of Operation

The agent filter's behavior when processing requests is based on the filter mode of operation.

The agent filter mode can be set either globally, which applies to all context paths protected by the agent, or on a per-context path level, overriding the global setting.

The filter mode can be set to one of the following values:

Agent Filter Modes

Filter Mode	Requires Authentication	Requires Authorization?	Comments
`URL_POLICY`	Yes	Yes	AM performs the following tasks: • Issues an OIDC JWT to the client after successful authentication [a] • Checks resource-based policies to evaluate whether the client can access the resource [b]
`SSO_ONLY`	Yes	No	AM issues an OIDC JWT to the client after successful authentication.
`NONE`	No	No	This mode disables the agent filter from taking any action on incoming requests. If logging is enabled, the agent filter logs all incoming requests for auditing purposes.
`ALL`	Yes	Yes	This mode behaves in the same way as the `URL_POLICY` mode and is kept for backward-compatibility purposes.
`J2EE_POLICY`	-	-	*This mode does not apply to Java Agents 5.6*, but it shows in the AM 5.5 agent profile page for backward-compatibility purposes.

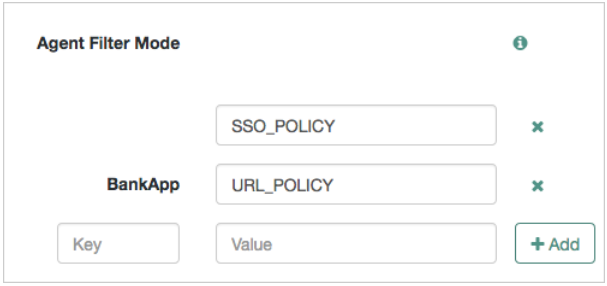
[a] For more information about AM authentication mechanisms, see *ForgeRock Access Management Authentication and Single Sign-On Guide*.
[b] For more information about AM policies, see *ForgeRock Access Management Authorization Guide*.

If neither the global or per-context paths filter mode are specified, the agent uses the default value, `URL_POLICY`.

To Configure the Agent Filter Mode

1. Navigate to Realms > Realm Name > Applications > Agents > Java > *Agent Name*.
2. On the Global tab, change the mode in the Agent Filter Mode (`com.sun.identity.agents.config.filter.mode`) property:
   - To set the global filter mode, enter the mode name in the Value field, for example `SSO_ONLY`, and then click Add.
   - To override the filter mode for a particular context path, enter the name of the context path in the Key field, for example `BankApp`, enter the mode name in the Value field, for example `URL_POLICY`, and then click Add.

Setting the Agent Filter Mode



3. Save your changes.

Configuring Audit Logging

Java agents support logging audit events for security, troubleshooting, and regulatory compliance. You can store agent audit event logs in the following ways:

- **Remotely**. Log audit events to the audit event handler configured in the AM realm. In a site comprised of several AM servers, Java agents write audit logs to the AM server that satisfies the agent's request for client authentication or resource authorization.

  Java agents cannot log audit events remotely if:

  - AM's Audit Logging Service is disabled.
  - No audit event handler is configured in the realm where the agent is configured.
  - All audit event handlers configured in the realm where the agent is configured are disabled.

  For more information about audit logging in AM, see the chapter Setting Up Audit Logging in the *ForgeRock Access Management Setup and Maintenance Guide*.

- **Locally**. Log audit events in JSON format to a file in the Java agent installation directory, `/java_agents/agent_type/logs/audit/`.

- **Locally and remotely**. Log audit events:

  - To a file in the agent installation directory.
  - To the audit event handler configured in the AM realm in which the agent profile is configured.

The following is an example of an agent log record:

```
{
   "timestamp":"2017-10-30T11:56:57Z",
   "eventName":"AM-ACCESS-OUTCOME",
```

```
   "transactionId":"608831c4-7351-4277-8a5f-b1a83fe2277e",
   "userId":"id=demo,ou=user,dc=openam,dc=forgerock,dc=org",
   "trackingIds":[
      "fd5c8ccf-7d97-49ba-a775-76c3c06eb933-82095",
      "fd5c8ccf-7d97-49ba-a775-76c3c06eb933-82177"
   ],
   "component":"Java Policy Agent",
   "realm":"/",
   "server":{
      "ip":"127.0.0.1",
      "port":8020
   },
   "client":{
      "ip":"127.0.0.1",
      "port":55180
   },
   "request":{
      "protocol":"HTTP/1.1",
      "operation":"GET"
   },
   "http":{
      "request":{
         "secure":false,
         "method":"GET",
         "path":"http://my.example.com:8020/examples/",
         "headers":{
            "referer":[
               "https://openam.example.com:8443/openam/oauth2/authorize?scope[...]"
            ],
            "accept-language":[
               "en,en-US;q=0.8,da;q=0.6,fr;q=0.4"
            ],
            "host":[
               "my.example.com:8020"
            ],
            "upgrade-insecure-requests":[
               "1"
            ],
            "connection":[
               "keep-alive"
            ],
            "cache-control":[
               "max-age=0"
            ],
            "accept-encoding":[
               "gzip, deflate"
            ],
            "user-agent":[
               "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
[...]"
            ],
            "accept":[
               "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8"
            ]
         },
         "cookies":{
            "am-auth-jwt":"eyJ0eXAiOiJKV1QiLCJhbGciOi[...]"
            "i18next":"en",
            "amlbcookie":"01",
```

```
            "iPlanetDirectoryPro":"Ts2zDkGUqgtkoxR[...]"
         }
      }
   },
   "response":{
      "status":"DENIED"
   },
   "_id":"fd5c8ccf-7d97-49ba-a775-76c3c06eb933-81703"
}
```

Note

Local audit logs do not have an `_id` attribute, which is an internal AM id.

The audit log format adheres to the log structure shared across the ForgeRock Identity Platform. For more information about the audit log format, see the section Audit Logging File Format in the *ForgeRock Access Management Setup and Maintenance Guide*.

Java agents support propagation of the transaction ID across the ForgeRock platform using the HTTP header `X-ForgeRock-TransactionId`. For more information about configuring the header, see *ForgeRock Access Management Setup and Maintenance Guide*.

By default, Java agents do not write audit log records. To configure audit logging, perform the following procedure:

To Configure Audit Logging

The procedure assumes the agent uses centralized configuration. Property names are also provided for local configuration agents.

1. In the AM console, navigate to Realms > *Realm Name* > Applications > Agents > *Java* > *Agent Name* > Global > Audit.

2. In the Audit Access Type property (`com.sun.identity.agents.config.audit.accesstype`), select the type of messages to log. For example, select `LOG_ALL` to log access allowed and access denied events.

3. In the Audit Log Location property (`com.sun.identity.agents.config.log.disposition`), select whether to write the audit logs locally to the agent installation (`LOCAL`), remotely to AM (`REMOTE`), or to both places (`ALL`). For example, keep `REMOTE` to log audit events to the AM instances.

4. (Optional) If you chose to log audit messages locally, enable the Rotate Local Audit Log property (`com.sun.identity.agents.config.local.log.rotate`) to rotate the audit log files upon reaching a maximum size.

5. (Optional) If you enabled the Rotate Local Audit Log property (`com.sun.identity.agents.config.local.log.size`), specify the maximum size of the audit log files in the Local Audit Log Rotation Size property.

Configuring Performance Monitoring

This section covers how to monitor the performance of Java agents.

You can monitor the performance of agents through the following interfaces:

Prometheus Monitoring

Prometheus is third-party software used for gathering and processing monitoring data.

For information about installing and running Prometheus, see the Prometheus documentation.

You can configure Java agents to expose an endpoint which Prometheus scrapes to obtain performance metrics from your protected applications.

Configure Prometheus to monitor the metrics endpoint exposed by the agent by using the `prometheus.yml` configuration file. For more information on configuring Prometheus, see the Prometheus configuration documentation.

> **Tip**
>
> Prometheus provides monitoring and processing for the information provided by Java agents, but further analysis and visualization may be desired. In this case, you can use tools such as Grafana to create customized charts and graphs based on the information collected by Prometheus.
>
> Example Grafana dashboards can be downloaded from the ForgeRock BackStage website.
>
> For more information on installing and running Grafana, see the Grafana website.

For information on enabling Prometheus monitoring, see "To Expose an Endpoint for Common REST and Prometheus Metrics".

ForgeRock® Common REST Monitoring

You can configure Java agents to expose an endpoint that allows REST clients to gather metrics about your protected applications, in JSON format.

For information on enabling Common REST monitoring, see "To Expose an Endpoint for Common REST and Prometheus Metrics".

CSV File-based Monitoring

You can write the metrics to comma-separated value (CSV) files, without having to expose an endpoint.

When enabled, the monitoring `.csv` files are written the same directory as the agent instance debug files, for example in `/path/to/java_agents/tomcat_agent/Agent_001/logs/debug/`.

For information on enabling CSV monitoring, see "To Enable Saving Metrics to CSV Files".

To Expose an Endpoint for Common REST and Prometheus Metrics

Common REST and Prometheus performance metrics are provided by an endpoint configured in the protected application's `web.xml` file. The endpoint must be accessible to the REST client or Prometheus server that will be making use of the performance data.

To configure an agent instance to expose the endpoint for metrics, perform the following steps:

1. For each protected application that will expose metrics, edit the application's `web.xml` file.

   The following Tomcat example exposes a base endpoint named `/metrics`:

   ```
   <servlet>
      <servlet-name>AgentMonitoring</servlet-name>
      <servlet-class>org.forgerock.http.servlet.HttpFrameworkServlet</servlet-class>
      <init-param>
          <param-name>application-loader</param-name>
          <param-value>guice</param-value>
      </init-param>
   </servlet>
   <servlet-mapping>
       <servlet-name>AgentMonitoring</servlet-name>
       <url-pattern>/metrics/*</url-pattern>
   </servlet-mapping>
   ```

   You can choose any name for the exposed base endpoint, but you must ensure it does not conflict with any of the builtin agent endpoints, for example `/sunwCDSSORedirectURI`.

2. Allow access to the base endpoint used for monitoring applications protected by the agent by using one of the following methods:

   - Create a Not Enforced URI rule for the base endpoint.

     For example:

     ```
     */metrics/*
     ```

     Note that this would allow open access to the metrics base endpoint.

     For more information, see Not-Enforced URI Processing Properties.

   - Create a Compound Not-Enforced URI and IP rule for the base endpoint.

     A Compound Not-Enforced URI and IP rule can allow access from only the IP addresses of the REST clients or Prometheus server.

     For example, the following rule allows access to the `/metrics` endpoint to HTTP requests that come from the IP address range from 192.168.1.1 to 192.168.1.3:

     ```
     192.168.1.1-192.168.1.3 | */metrics/*
     ```

HTTP requests from other IP addresses would not be able to access the metrics base endpoint.

For more information, see Not-Enforced IP Processing Properties.

- Create an authorization policy in AM to restrict access to the metrics base endpoint.

  Note that the metric base endpoint does not require login credentials. You can use a policy to ensure that requests to the endpoints are authenticated against the AM instance.

  For more information, see Configuring Policies in the *ForgeRock Access Management 6.5 Authorization Guide*.

3. The Common REST performance monitoring endpoint will now be available under the path used by the protected application, for example `https://mydomain.example.com/myapp/metrics/crest`.

   Configure your REST clients to access the endpoint to gather performance metric data. Ensure you include the relevant credentials if you are protecting the endpoint by using policies in AM.

4. (Optional) The Prometheus performance monitoring endpoint is available under the path used by the protected application, for example `https://mydomain.example.com/myapp/metrics/prometheus`.

   Configure your Prometheus server to access the endpoint to gather performance metric data. Ensure you include the relevant credentials if you are protecting the endpoint by using policies in AM.

To Enable Saving Metrics to CSV Files

- Writing monitoring metrics to CSV files is enabled by setting the `org.forgerock.agents.config.monitoring.to.csv` property:

  - To configure the agent to write metric information to CSV files, set the `org.forgerock.agents.config.monitoring.to.csv` property to `true`.

  - To prevent the agent writing metric information to CSV files, set the `org.forgerock.agents.config.monitoring.to.csv` property to `false`.

  For information on where to set agent properties, see "Configuration Location".

For reference information on Java Agent performance metrics, see "Monitoring Reference".

Configuring Java Agents for SSL Communication

For security reasons, your environment may require that your Java agents communicate with AM over SSL. To configure the agents, perform the steps in the following procedure:

To Configure Java Agents for SSL Communication

1. Import a CA certificate in the JDK truststore, usually `$JAVA_HOME/jre/lib/security/cacerts`. The certificate should be either the same one configured for SSL purposes in the container where AM is installed, or one signed with the same CA root certificate. For example:

   ```
   $ keytool \
   -import \
   -trustcacerts \
   -alias agentcert \
   -file /path/to/cacert.pem \
   -keystore $JAVA_HOME/jre/lib/security/cacerts
   ```

   Ensure all containers where AM is installed trust the certificate stored in the JDK truststore, and that the JDK trusts the certificates stored on the containers where AM is installed.

2. Edit the `/path/to/java_agents/agent_type/agent_instance/config/OpenSSOAgentBootstrap.properties` file and add the following properties:

   - `javax.net.ssl.trustStore`. Specifies the full path to the JDK truststore.
   - `javax.net.ssl.trustStorePassword`. Specifies the password of the truststore.

   For example:

   ```
   javax.net.ssl.trustStore=/Library/Java/JavaVirtualMachines/jdk1.8.0_101.jdk/Contents/Home/jre/lib/
   security/cacerts
   javax.net.ssl.trustStorePassword=changeit
   ```

   Note

   For backward-compatibility purposes, you can also provide the truststore and the password to the agent by specifying them as Java properties in the container's start-up sequence. For example, add them to Tomcat's **`$CATALINA_OPS`** variable instead of specifying them in the `OpenSSOAgentBootstrap.properties` file:

   ```
   $ export CATALINA_OPTS="$CATALINA_OPTS \
   -Djavax.net.ssl.trustStore=$JAVA_HOME/jre/lib/security/cacerts \
   -Djavax.net.ssl.trustStorePassword=changeit"
   ```

3. Restart the Java agent.

Supporting Load Balancers and Reverse Proxies Between Clients and Agents

When your environment has reverse proxies or load balancers configured between the agents and the clients, you must perform additional configuration in the agents to account for the anonymization of both the clients and the agents.

Failure to do so may cause policy evaluation and other agent features to fail.

For more information, see "*Configuring Environments With Load Balancers and Reverse Proxies*".

Chapter 6
Upgrading Java Agents

The process of upgrading a Java agent consist of uninstalling the old agent and installing a new one. There is no requirement to create a new agent profile.

To upgrade Java agents, perform the following procedure:

To Upgrade Java Agents

1. Refer to the Release Notes for information about changes in support and functionality.
2. Back up the agent installation and the application container configuration directories. For example:

   ```
   $ cp -r /path/to/java_agents/tomcat_v7_agent /path/to/backup
   $ cp -r /path/to/tomcat/webapps/agentapp /path/to/backup
   ```

   If the configuration is stored centrally in AM, back it up as described in the *ForgeRock Access Management Maintenance Guide*.
3. Redirect client traffic away from the protected application.
4. Stop the web application container where the Java agent is installed.
5. Remove the old Java agent.

   For example, to remove an old Tomcat Java agent, see "Removing the Tomcat Java Agent". If the uninstall process has changed, refer to the version of the *Java Agent Guide* that corresponds to your Java agent.
6. Install the new agent.

   For example, to install a Tomcat Java agent, see "Installing the Tomcat Java Agent".

   The installer creates new `OpenSSOAgentConfiguration.properties` and `OpenSSOAgentBootstrap.properties` files containing adequate properties for the particular agent version.
7. Review the agent configuration:

   - If the agent configuration is stored in the AM configuration store, review the Release Notes and the ForgeRock Access Management Release Notes to check what is new and possible changes to AM and the agent. Then, adjust the agent configuration if required using the AM console.

- If the agent configuration is stored locally, review the `OpenSSOAgentConfiguration.properties` file. Use the backed-up copy of the configuration file for guidance, and the Release Notes and the ForgeRock Access Management Release Notes to check what is new and possible changes to AM and the agent. Then, update the file manually to contain the properties required for your environment.

  The `OpenSSOAgentBootstrap.properties` file created by the installer already contain bootstrap properties relevant to the new version of the agent.

8. Ensure the communication between AM and the Java agent is secured with the appropriate keys. For more information, see "Configuring Access Management Servers to Communicate With Java Agents".
9. Start the web application container where the agent is installed.
10. Validate that the Java agent is performing as expected.

    For example, navigate to a protected page on the web site and confirm whether you can access it according to your configuration.
11. Allow client traffic to flow to the protected application.

Chapter 7

Removing Java Agents

The following table contains a list of sections containing information about removing Java agents on supported platforms:

Task	Section
Remove Java agents on Apache Tomcat	Section
Remove Java agents on Red Hat JBoss	Section
Remove Java agents on Eclipse Jetty	Section
Remove Java agents on Oracle WebLogic	Section
Remove Java agents on IBM WebSphere	Section

Removing the Tomcat Java Agent

Complete the following procedure to remove the Tomcat Java agent:

To Remove the Tomcat Java Agent

1. Shut down the Tomcat server where the agent is installed.
2. Run the **agentadmin** command with the `--listAgents` option to output a list of installed agent instances. For example:

   ```
   $ agentadmin --listAgents

   The following agents are configured on this Application Server.

   The following are the details for agent Agent_001 :-
   Tomcat Server Config Directory: /path/to/apache-tomcat/conf
   ```

   Make a note of the agent configuration details of the instance you want to remove.
3. Run the **agentadmin** command with the `--uninstall` option.

   ```
   $ agentadmin --uninstall
   ```

   a. Enter the path of the Tomcat installation directory. For example, `/path/to/apache-tomcat/conf`.

```
Enter the complete path to the directory which is used by Tomcat Server to
store its configuration Files. This directory uniquely identifies the
Tomcat Server instance that is secured by this Agent.
[ ? : Help, ! : Exit ]
Enter the Tomcat Server Config Directory Path
[/opt/apache-tomcat-6.0.14/conf]: /path/to/apache-tomcat/conf
```

b. Review a summary of your responses and select an action to continue: uninstall, go back a step, start over, or exit:

```
 ---------------------------------------------
SUMMARY OF YOUR RESPONSES
---------------------------------------------
Tomcat Server Config Directory : /path/to/apache-tomcat/conf

Verify your settings above and decide from the choices below.
1. Continue with Uninstallation
2. Back to the last interaction
3. Start Over
4. Exit
Please make your selection [1]: 1
DONE.

Removing the Agent jar/locale files from the classloader directory ...DONE.

Deleting the config directory
/path/to/java_agents/tomcat_agent/Agent_001/config
...DONE.

Removing OpenAM Tomcat Agent Realm from Server XML file :
/path/to/apache-tomcat/conf/server.xml ...DONE.

Removing filter from Global deployment descriptor file :
/path/to/apache-tomcat/conf/web.xml ...DONE.

Removing OpenAM Tomcat Agent Filter and Form login authentication from Web
applications ...DONE.


Uninstall log file location:
/path/to/java_agents/tomcat_agent/installer-logs/audit/uninstall.log

Thank you for using AM Policy Agent
```

Removing the JBoss Java Agent

Complete the following procedure to remove the JBoss Java agent:

To Remove the JBoss Java Agent

1. Shut down the JBoss server where the agent is installed.

2. Run the **agentadmin** command with the `--listAgents` option to output a list of installed agent instances. For example:

```
$ agentadmin --listAgents
The following agents are configured on this Application Server.

The following are the details for agent Agent_001 :-
JBoss home directory: /path/to/jboss
JBoss domain profile name: null
JBoss deployment mode: standalone
```

   Make a note of the agent configuration details of the instance you want to remove.

3. Run the **agentadmin** command with the `--uninstall` option.

```
$ agentadmin --uninstall
```

   a. Enter the path to the JBoss installation directory. For example, `/path/to/jboss`.

```
Enter the complete path to the home directory of the JBoss instance.
[ ? : Help, ! : Exit ]
Enter the path to the JBoss installation: /path/to/jboss
```

   b. Enter the deployment mode of the JBoss installation to uninstall. Possible values are `domain` or `standalone`.

```
Enter the name of the deployment mode of the JBoss installation that you wish
to use with this agent. Supported values are: domain, standalone.
[ ? : Help, < : Back, ! : Exit ]
Enter the deployment mode of JBoss [standalone]: standalone
```

   c. Review a summary of your responses and select an action to continue: uninstall, go back a step, start over, or exit:

```
-----------------------------------------------
SUMMARY OF YOUR RESPONSES
-----------------------------------------------
JBoss home directory : /path/to/jboss
JBoss deployment mode : standalone

Verify your settings above and decide from the choices below.
1. Continue with Uninstallation
2. Back to the last interaction
3. Start Over
4. Exit
Please make your selection [1]: 1

Removing Agent settings from
/path/to/jboss/standalone/configuration/standalone.xml
file ...DONE.
DONE.
DONE.

Deleting the config directory
/path/to/java_agents/jboss_agent/Agent_001/config ...DONE.


Uninstall log file location:
/path/to/java_agents/jboss_agent/installer-logs/audit/uninstall.log

Thank you for using AM Policy Agent.
```

Removing the Jetty Java Agent

Complete the following procedure to remove the Jetty Java agent:

To Remove the Jetty Java Agent

1. Shut down the Jetty server where the agent is installed.
2. Run the **agentadmin** command with the `--listAgents` options to output a list of installed agent instances. For example:

   ```
   $ ./agentadmin --listAgents

   The following agents are configured on this Application Server.

   The following are the details for agent Agent_001 :-
   Jetty Server Config Directory:
   /path/to/jetty/etc
   ```

   Make a note of the agent configuration details of the instance you want to remove.

   a. Run the **agentadmin** command with the `--uninstall` option.

      ```
      $ agentadmin --uninstall
      ```

b. Enter the path of the Jetty configuration directory. For example, `/path/to/jetty/etc`.

```
Enter the complete path to the directory which is used by Jetty Server to store
its configuration Files. This directory uniquely identifies the Jetty
Server instance that is secured by this Agent.
[ ? : Help, ! : Exit ]
Enter the Jetty Server Config Directory Path [/opt/jetty/etc]: /path/to/jetty/etc
```

c. Review a summary of your responses and select an action to continue: uninstall, go back a step, start over, or exit:

```
------------------------------------------------
SUMMARY OF YOUR RESPONSES
------------------------------------------------
Jetty Server Config Directory :
/path/to/jetty/


Verify your settings above and decide from the choices below.
1. Continue with Uninstallation
2. Back to the last interaction
3. Start Over
4. Exit
Please make your selection [1]: 1


Removing the agent classpath from start.conf file ...DONE.

Deleting the config directory
/path/to/java_agents/jetty_agent/Agent_001/config
...DONE.

Removing Login configuration files: amlogin.conf amlogin.xml...DONE.

Removing Agent app...DONE.


Uninstall log file location:
/path/to/java_agents/jetty_agent/installer-logs/audit/uninstall.log

Thank you for using AM Policy Agent
```

Removing the WebLogic Java Agent

Complete the following procedure to remove the WebLogic Java agent:

To Remove the WebLogic Java Agent

1. Shut down the WebLogic server where the agent is installed.
2. Run the **agentadmin** with the `--listAgents` option to output a list of installed agent instances. For example:

```
The following agents are configured on this Application Server.

The following are the details for agent Agent_001 :-
WebLogic Server instance name: AdminServer
Startup script location:
/Oracle_Home/user_projects/domains/base_domain/startWebLogic.sh
```

Make a note of the agent configuration details of the instance you want to remove.

3. Run the **agentadmin** with the `--uninstall` option.

```
$ agentadmin --uninstall
```

   a. Enter the path to the `startWebLogic.sh` file of the WebLogic domain where you want to install the agent. For example, `/Oracle_Home/user_projects/domains/base_domain/startWebLogic.sh`.

```
Enter the path to the location of the script used to start the WebLogic domain.
Please ensure that the agent is first installed on the admin server instance
before installing on any managed server instance.
[ ? : Help, ! : Exit ]
Enter the Startup script location
[/usr/local/bea/user_projects/domains/base_domain/startWebLogic.sh]: /Oracle_Home/user_projects/
domains/base_domain/startWebLogic.sh
```

   b. Enter the name of the WebLogic instance. For example, `AdminServer`.

```
Enter the name of the WebLogic Server instance secured by the agent.
[ ? : Help, < : Back, ! : Exit ]
Enter the WebLogic Server instance name [AdminServer]: AdminServer
```

   c. Review a summary of your responses and select an action to continue: uninstall, go back a step, start over, or exit:

```
------------------------------------------------
SUMMARY OF YOUR RESPONSES
------------------------------------------------
Startup script location :
/path/to/weblogic/mydomain/startWebLogic.sh
WebLogic Server instance name : AdminServer

Verify your settings above and decide from the choices below.
1. Continue with Uninstallation
2. Back to the last interaction
3. Start Over
4. Exit
Please make your selection [1]: 1

Remove amauthprovider.jar from
/path/to/weblogic/server/lib/mbeantypes
...DONE.

Deleting the config directory
/path/to/java_agents/weblogic_vs_agent/Agent_001/config
...DONE.

UnConfigure
/path/to/weblogic/mydomain/setAgentEnv_AdminServer.sh
...DONE.


Uninstall log file location:
/path/to/java_agents/weblogic_vs_agent/installer-logs/audit/uninstall.log

Thank you for using AM Policy Agent
```

Removing the WebSphere Java Agent

Complete the following procedure to remove the WebSphere Java Agent:

To Remove the WebSphere Java Agent

1. Shut down the WebSphere server where the agent is installed.
2. Run the **agentadmin** command with the `--listAgents` option to output a list of installed agent instances. For example:

   ```
   The following agents are configured on this Application Server.

   The following are the details for agent Agent_001 :-
   WebSphere Install Root Directory: /path/to/WebSphere/AppServer
   Instance Server name: server1
   Instance Config Directory:
   /path/to/WebSphere/AppServer/profiles/AppServ01/config/cells/DefaultCell01/nodes/DefaultNode01/
   servers/server1
   ```

   Make a note of the agent configuration details of the instance you want to remove.

3. Run the **agentadmin** command with the `--uninstall` option:

```
$ agentadmin --uninstall
```

   a. Enter the path to the configuration directory of the server instance for the WebSphere node. For example, `/path/to/WebSphere/AppServer/profiles/AppServ01/config/cells/DefaultCell01/nodes/DefaultNode01/servers/server1`.

```
Enter the fully qualified path to the configuration directory of the Server
Instance for the WebSphere node.
[ ? : Help, ! : Exit ]
Enter the Instance Config Directory
[/opt/IBM/WebSphere/AppServer/profiles/AppSrv01/config/cells/<hostname>Node01Cell/nodes/
<hostname>Node01/servers/server1]: /path/to/WebSphere/AppServer/profiles/AppServ01/config/cells/
DefaultCell01/nodes/DefaultNode01/servers/server1
```

   b. Enter the name of the server instance where the agent will be removed. For example, `server1`.

```
Enter the Server Instance name.
[ ? : Help, < : Back, ! : Exit ]
Enter the Server Instance name [server1]: server1
```

   c. Enter the path to the WebSphere install directory. For example, `/path/to/WebSphere/AppServer`.

```
Enter the WebSphere Install Root directory.
[ ? : Help, < : Back, ! : Exit ]
Enter the WebSphere Install Root directory
[/opt/IBM/WebSphere/AppServer]: /path/to/WebSphere/AppServer
```

   d. Review a summary of your responses and select an action to continue: uninstall, go back a step, start over, or exit:

```
 -----------------------------------------------
SUMMARY OF YOUR RESPONSES
-----------------------------------------------
Instance Config Directory :
/path/to/WebSphere/AppServer/profiles/AppServ01/config/cells/DefaultCell01/nodes/DefaultNode01/
servers/server1

Instance Server name : server1
WebSphere Install Root Directory : /path/to/WebSphere/AppServer

Verify your settings above and decide from the choices below.
1. Continue with Uninstallation
2. Back to the last interaction
3. Start Over
4. Exit
Please make your selection [1]: 1
Remove jars from /path/to/WebSphere/AppServer/lib/ext...DONE.

Deleting the config directory
/path/to/java_agents/websphere_agent/Agent_001/config ...DONE.

Unconfigure server.xml file
/path/to/WebSphere/AppServer/profiles/AppServ01/config/cells/DefaultCell01/nodes/DefaultNode01/
servers/server1/server.xml
...DONE.


Uninstall log file location:
/path/to/java_agents/websphere_agent/installer-logs/audit/uninstall.log

Thank you for using AM Policy Agent
```

Chapter 8
Troubleshooting

This chapter offers solutions to issues that may occur during installation of AM Java agents.

Solutions to Common Issues

Q: I am trying to install a Java agent, connecting to AM over HTTPS, and seeing the following error:

```
AM server URL: https://openam.example.com:8443/openam

WARNING: Unable to connect to OpenAM server URL. Please specify the
correct OpenAM server URL by hitting the Back button (<) or if the OpenAM
server URL is not started and you want to start it later, please proceed with
the installation.
If OpenAM server is SSL enabled and the root CA certificate for the OpenAM
server certificate has been not imported into installer JVMs key store (see
installer-logs/debug/Agent.log for detailed exception), import the root
CA certificate and restart the installer; or continue installation without
verifying OpenAM server URL.
```

What should I do?

A: The Java platform includes certificates from many certificate authorities (CAs). If, however, you run your own CA, or you use self-signed certificates for HTTPS on the web application container where you run AM, then the **agentadmin** command cannot trust the certificate presented during connection to AM, and so cannot complete installation correctly.

After setting up the web application container where you run AM to use HTTPS, get the certificate to trust in a certificate file. The certificate you want is that of the CA who signed the container certificate, or the certificate itself if the container certificate is self-signed.

Copy the certificate file to the system where you plan to install the Java agent. Import the certificate into a trust store that you will use during Java agent installation. If you import the certificate into the default trust store for the Java platform, then the **agentadmin** command can recognize it without additional configuration.

Export and import of self-signed certificates is demonstrated in the *ForgeRock Access Management Install Guide* section *Using Self-Signed Certificates*.

Q: I am trying to install the WebSphere Java agent on Linux. The system has IBM Java. When I run **agentadmin --install**, the script fails to encrypt the password from the password file, ending with this message:

```
ERROR: An unknown error has occurred (null). Please try again.
```

What should I do?

A: You must edit **agentadmin** to use IBMJCE, and then try again.

See "To Install With IBM Java".

Q: I have client-based (stateless) sessions configured in AM, and I am getting infinite redirection loops. In the `debug.log` file I can see messages similar to the following:

```
2018-03-15 16:23:10.538 +0000 ERROR [c5319caa-beeb-5a44-a098-d5575e768348]state identifier not
 present in authentication state
2018-03-15 16:23:10.538 +0000 WARNING [c5319caa-beeb-5a44-a098-d5575e768348]unable to verify pre-
authentication cookie
2018-03-15 16:23:10.538 +0000 WARNING [c5319caa-beeb-5a44-a098-
d5575e768348]convert_request_after_authn_post(): unable to retrieve pre-authentication request data
2018-03-15 16:23:10.538 +0000 DEBUG [c5319caa-beeb-5a44-a098-d5575e768348] exit status: forbidden
 (3), HTTP status: 403, subrequest 0
```

What is happening?

A: In this case, the redirection loop happens because the client-based (stateless) session cookie is surpassing the maximum supported browser header size. Since the cookie is incomplete, AM cannot validate it.

To ensure the session cookie does not surpass the browser supported size, configure either signing and compression or encryption and compression.

For more information, see the *ForgeRock Access Management Authentication and Single Sign-On Guide*.

Q: I have upgraded my agent and I see the following message in the Java agent log:

```
redirect_uri_mismatch. The redirection URI provided does not match a pre-registered value.
```

What should I do?

A: Java agents 5.6 only accept requests sent to the URL specified by the Agent Root URL for CDSSO property. For example, `http://agent.example.com:8080/`.

As a security measure, Java agents prevent you from accessing the agent on URLs not defined in the Agent Root URL for CDSSO property. Add entries to this property when:

- Configuring the Alternative Agent Protocol (`com.sun.identity.agents.config.agent.protocol`) property to access the agent through different protocols. For example, `http://agent.example.com/` and `https://agent.example.com/`.
- Configuring the Alternative Agent Host Name (`com.sun.identity.agents.config.agent.host`) property to access the agent through different virtual host names. For example, `http://agent.example.com/` and `http://internal.example.com/`.

- Configuring the Alternative Agent Port Name (`com.sun.identity.agents.config.agent.port`) property to access the agent through different ports. For example, `http://agent.example.com/` and `http://agent.example.com:8080/`.

Q: After installing a Java agent on WebSphere, accessing a URL for a folder in a protected application such as `http://openam.example.com:9080/test/` results in `Error 404: SRVE0190E: File not found: {0}`, and redirection fails. What should I do to work around this problem?

A: Perform the following steps to work around the problem, by setting the WebSphere custom property `com.ibm.ws.webcontainer.invokeFiltersCompatibility=true`:

1. In the WebSphere administrative console, browse to Servers > Server Types, and then click WebSphere application servers.
2. Click the server to apply the custom property to.
3. Navigate to Configuration > Container settings > Web Container Settings > Web container.
4. Under Configuration > Additional Properties, click Custom Properties.
5. In the Custom Properties page, click New.
6. In the settings page, enter the Name `com.ibm.ws.webcontainer.invokeFiltersCompatibility` and Value `true` for the custom property.

   Some properties are case-sensitive.
7. Click Apply or OK as applicable.
8. Click Save in the Message box that appears.
9. Restart the server for the custom property to take effect.

See the IBM documentation on *Setting webcontainer custom properties* for additional information.

Chapter 9
Reference

This reference section covers Java agent and agent authenticator configuration properties.

Configuring Java Agent Properties

When you create the agent profile, you can choose whether to store the agent configuration in AM's configuration store or locally to the agent installation [1]. This section covers centralized configuration, indicating the corresponding properties for use in a local configuration file where applicable. [2]

After changing properties specified as "Hot-swap: no", you must restart the container where the Java agent is installed for the changes to take effect.

Configuring Bootstrap Properties

These properties are set in the `config/OpenSSOAgentBootstrap.properties` file.

`am.encryption.pwd`

When using an encrypted password, set this to the encryption key used to encrypt the agent profile password.

~~**`com.iplanet.am.naming.url`**~~

This property does not apply to Java Agents 5.6.

`com.iplanet.am.service.secret`

When using a plain text password, set this to the password for the agent profile, and leave `am.encryption.pwd` blank.

When using an encrypted password, set this to the encrypted version of the password for the agent profile. Use the command **./agentadmin --encrypt *agentInstance passwordFile*** to get the encrypted version.

Default: not set

[1] See "Configuration Location" for more information about the agent configuration.
[2] The configuration file syntax is the same as of a standard Java properties file. See java.util.Properties.load() for a description of the format.

`org.forgerock.agents.public.am.url`

Specifies the full URL of AM when it is behind a proxy during the custom login redirection flow. For example, `protocol://public_am_fqdn:port/openam`.

Use this property both of the following points are true:

- Your environment uses custom login pages (non-OIDC-compliant flows), and the custom login pages are not in the same domain as the agent.
- Your custom login pages are in a network that can only access AM using a proxy, a firewall, or any other technology that remaps the AM URL to one accessible by the custom login pages.

Consider an example where the traffic between AM and the agent happens through the *example-internal.com* network, but the custom login pages are on the *example-external.com* domain. The traffic between the custom pages and AM is translated from `am.example-internal.com` into `am.example-external.com`.

You would configure *https://am.example-external.com:8443/openam* as the public AM URL.

The default value is a combination of the values of the `com.iplanet.am.server.host`, `com.iplanet.am.server.port`, `com.iplanet.am.server.protocol`, and `com.iplanet.am.services.deploymentDescriptor` properties.

`com.iplanet.am.services.deploymentDescriptor`

Specifies the URI under which AM is deployed, such as `/openam`.

Default: not set

`com.iplanet.services.debug.directory`

Specifies the full path of the directory where the Java agent writes debug log files.

Default: *`/path/to/agent/agent_type/agent_instance`*`/logs/debug`

`com.sun.identity.agents.app.username`

Specifies the agent profile name.

Default: not set

`com.sun.identity.agents.config.local.logfile`

Specifies the full path of the Java agent's audit log file.

Default: *`/path/to/agent/agent_type/agent_instance`*`/logs/audit/amAgent_`*`AM_FQDN_PORT`*`.log`

`com.sun.identity.agents.config.lock.enable`

When `true`, specifies that an agent restart is required to allow agent configuration changes, even for hot-swappable parameters.

Default: `false`

`com.sun.identity.agents.config.organization.name`

Specifies the realm name where the agent authenticates to AM.

Default: `/` (top-level realm)

`com.sun.identity.agents.config.profilename`

Specifies the profile name used to fetch agent configuration data. Unless multiple Java agents use the same credentials to authenticate, this is the same as `com.sun.identity.agents.app.username`.

Default: not set

~~`com.sun.identity.agents.config.service.resolver`~~

This property does not apply to Java Agents 5.6.

`com.sun.services.debug.mergeall`

When `on`, the Java agent writes all debug messages to a single file under `com.iplanet.services.debug.directory`.

Default: `on`

`javax.net.ssl.trustStore`

Specifies the full path to the JVM truststore. Use this property to enable SSL communication with AM.

For more information, see "Configuring Java Agents for SSL Communication".

Default: not set

`javax.net.ssl.trustStorePassword`

Specifies the password of the truststore defined by the `javax.net.ssl.trustStore` property.

For more information, see "Configuring Java Agents for SSL Communication".

Default: not set

`org.forgerock.agents.prometheus.monitoring.enabled`

When `true` enables the Prometheus performance monitoring endpoint.

For more information, see "Configuring Bootstrap Properties".

Default: `true`

`org.forgerock.openam.url.connectTimeout`

Sets the TCP connection timeout for outbound HTTP connections created by the Java agent. file.

Default: not set

Configuring Global Properties

This section covers global Java agent properties. After creating the agent profile, access these properties in the AM console by navigating to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Global.

This section describes the following property groups:

- Profile Properties
- General Properties
- User Mapping Properties
- Audit Properties
- Fully Qualified Domain Name Checking Properties

Profile Properties

Group

For assigning the Java agent to a previously configured group in order to inherit selected properties from the group.

Password

Specifies the password used when creating the password file and when installing the Java agent.

If you change this password, you must modify manually the password of the bootstrap property `com.iplanet.am.service.secret`. For more information, see "Configuring Bootstrap Properties".

Status

Specifies the status of the agent configuration.

~~**Agent Notification URL**~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

When creating an agent profile, the AM console configures a default value for this property to maintain compatibility with earlier versions of the Java agent. The default value can be safely removed.

Property: `com.sun.identity.client.notification.url`

Location of Agent Configuration Repository

Specifies whether the Java agent configuration runs in *local* or *centralized* mode. For background information on these modes, see "Configuration Location".

To configure `local` mode, make sure the `com.sun.identity.agents.config.repository.location=local` is in the bootstrap properties file.

Note

At startup, the agent always reads the bootstrap properties file, and then the configuration properties file [3]. The agent's behavior then depends entirely on the value of the `com.sun.identity.agents.config.repository.location` property.

If the property is set to `LOCAL`, the agent will use all of the properties it has retrieved and continue working.

If the property is set to `CENTRALIZED` or is not defined at all, the agent will ignore all values from the configuration properties file, and while retaining the retrieved bootstrap properties, download its configuration from AM.

To revert to `centralized` mode, remove the `com.sun.identity.agents.config.repository.location` property in the bootstrap file, and then restart the agent's container.

Default: `centralized`

Property: `com.sun.identity.agents.config.repository.location`

Configuration Reload Interval

Specifies the time interval in seconds after which the Java agent reloads the agent profile. The behavior of this property is determined by the agent profile's configuration location:

- **Centralized configuration**. The Java agent reloads the configuration from AM.
- **Local configuration**. The Java agent reloads the configuration from the local files if the local files have been modified.

Tip

Notifications ensure that Java agents with centralized configuration reload the configuration when the administrator makes a change to a hot-swappable configuration property. Enable this property if notifications are disabled or if the Java agent stores its configuration locally.

Set the property to `0` to disable it.

[3] These files are `/path/to/java_agents/agent_type/agent_instance/config/OpenSSOAgentBootstrap.properties` and `/path/to/java_agents/agent_type/agent_instance/OpenSSOAgentConfiguration.properties`, respectively.

Default: `0`

Property: `com.sun.identity.agents.config.load.interval`

Hot-swap: yes

Agent Configuration Change Notification

Specifies whether AM sends a notification to the Java agent to reread the agent profile after a change to a hot-swappable property. This property only applies when you store the agent profile in AM's configuration data store.

Default: `true`

Property: `com.sun.identity.agents.config.change.notification.enable`

Hot-swap: no

Web Socket Connection Interval

Specifies the time in minutes after which Java agents reopen their WebSocket connection to AM. This property helps ensure a balanced distribution of connections across the AM servers on the site.

Default: `30`

Property: `org.forgerock.openam.agents.config.balance.websocket.connection.interval.in.minutes`

Hot-swap: yes

JWT Cookie Name

Specifies the name of the cookie that holds the OpenID Connect JSON web token (JWT) on the user's browser.

Before changing the name of this cookie, consider the following points:

- This cookie is only used by the Java agent and is never presented to AM.
- The name of this cookie must be unique across the set of cookies the user's browser receives, since some browsers behave in unexpected ways when receiving several cookies with the same name. For example, you should not set the JWT cookie name to `iPlanetDirectoryPro`, which is the default name of AM's session cookie.

Default: `am-auth-jwt`

Property: `org.forgerock.openam.agents.config.jwt.name`

Hot-swap: yes

JWT Cache Size (Not yet in the AM console)[4]

Specifies the maximum number of decoded OpenID Connect JWTs the Java agent stores in the cache. When the cache fills up, it evicts items on a last-accessed basis.

Default: `1000`

Property: `org.forgerock.openam.agents.config.jwt.cache.size`

JWT Cache Timeout (Not yet in the AM console)[4]

Specifies the time interval in minutes after which a JWT in the Java agent's cache expires.

Default: `30`

Property: `org.forgerock.openam.agents.config.jwt.cache.ttl.minutes`

Convert SSO Tokens into OpenID Connect JWTs (Not yet in the AM console)[4]

Specifies whether the agent should convert SSO tokens (`iPlanetDirectoryPro` cookies) into OpenID Connect JWTs, to make them compliant with the agent's default login redirection mode.

Set this property when your end users access resources protected by both Java Agents 3.5.x (which use SSO tokens) and 5.x (which use OpenID Connect JWTs). Converting the SSO token to a JWT will ensure a seamless experience to the user without additional redirection or re-authentication.

When this property is enabled, the agent makes a request to AM to exchange the SSO token for a JWT.

> **Tip**
>
> The client application is responsible for appending the JWT to subsequent calls to protected resources. Failure to do so will cause the agent to request additional JWTs from AM.

Default: `false`

Property: `com.forgerock.agents.accept.ipdp.cookie`

Hot-swap: yes

Exchanged SSO Token Cache Time to Live (Not yet in the AM console)[4]

Specifies how long to cache the results of exchanging an SSO token for a JWT, in minutes.

The returned JWT is cached against the relevant SSO token. If the same SSO token is presented in the future, but before the cache expires, the agent does not need to request a new JWT from AM. Instead, it retrieves the correct JWT from its cache.

[4]Set this property as a custom property in AM, by navigating to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

Default: `5`

Property: `org.forgerock.agents.sso.exchange.cache.ttl.minutes`

Exchanged SSO Token Cache Max Records (Not yet in the AM console)[4]

Specifies the maximum number of entries allowed when caching the results of exchanging an SSO token for a JWT, in minutes.

If the maximum number of records is reached, the oldest records in the cache are overwritten.

Default: `100`

Property: `org.forgerock.agents.sso.exchange.cache.size`

Agent Root URL for CDSSO

The Java agent root URL for CDSSO. The valid value is in the format *`protocol://hostname:port/`* where *protocol* represents the protocol used, such as `http` or `https`, *hostname* represents the host name of the system where the Java agent resides, and *port* represents the port number on which the Java agent is installed. The slash following the port number is required.

If the server where the Java agent is installed has virtual host names, add URLs with the virtual host names to this list as well. AM checks that `goto` URLs match one of the Java agent root URLs for CDSSO.

Default: *`agent-root-URL`*

Property: `sunIdentityServerDeviceKeyValue[`*`n`*`]`

General Properties

Agent Filter Mode

Specifies the agent filter's mode of operation. The mode can be set to one of the following values:

Agent Filter Modes

Filter Mode	Requires Authentication	Requires Authorization?	Comments
`URL_POLICY`	Yes	Yes	AM performs the following tasks: • Issues an OIDC JWT to the client after successful authentication [a] • Checks resource-based policies to evaluate whether the client can access the resource [b]
`SSO_ONLY`	Yes	No	AM issues an OIDC JWT to the client after successful authentication.

Filter Mode	Requires Authentication	Requires Authorization?	Comments
`NONE`	No	No	This mode disables the agent filter from taking any action on incoming requests. If logging is enabled, the agent filter logs all incoming requests for auditing purposes.
`ALL`	Yes	Yes	This mode behaves in the same way as the `URL_POLICY` mode and is kept for backward-compatibility purposes.
`J2EE_POLICY`	-	-	*This mode does not apply to Java Agents 5.6*, but it shows in the AM 5.5 agent profile page for backward-compatibility purposes.

[a] For more information about AM authentication mechanisms, see *ForgeRock Access Management Authentication and Single Sign-On Guide*.
[b] For more information about AM policies, see *ForgeRock Access Management Authorization Guide*.

For more information, see "Configuring the Agent Filter".

Default: `ALL`

Property: `com.sun.identity.agents.config.filter.mode`

Hot-swap: yes

Idle Timeout Window (Not yet in the AM console)[4]

Specifies the time interval, in minutes, the agent will wait before making a call to AM to refresh a the session's idle timeout.

Sessions in AM have an idle timeout after which they expire. In general, when users access protected resources through an agent, the agent requests a policy decision on behalf of that user, which resets the idle timeout.

When the agent does not need to reach out to AM frequently, for example, when policy evaluation is already cached, sessions may unexpectedly expire in AM due to idle timeout before the user has finished accessing the application.

Agents make one call per active user session at the end of the time interval, provided that the user is actively accessing the application or site. If the user does not access the application during the configured window interval time, the agent will not make the call to AM at the end of the interval. Eventually, if the user is inactive for enough time, AM will log them out when the session reaches its idle timeout.

Configuring the idle timeout window to a short value, such as one minute, achieves a good balance between making additional calls to AM and providing a good user experience.

Increase this value only if the performance impact of making an extra call to AM every minute is noticeable enough in your environment.

Default: 1

Property: `org.forgerock.agents.idle.time.window.minutes`

Hot-swap: yes

HTTP Session Binding

When enabled, the Java agent invalidates the HTTP session upon login failure, when the user has no SSO session, or when the principal user name does not match the SSO user name.

Default: `true`

Property: `com.sun.identity.agents.config.httpsession.binding`

Login Attempt Limit

When set to a value other than zero, this defines the maximum number of failed login attempts allowed during a single browser session, after which the Java agent blocks requests from the user.

Default: `0`

Property: `com.sun.identity.agents.config.login.attempt.limit`

Custom Response Header

Specifies the custom headers the Java agent sets for the client. The key is the header name. The value is the header value. For example, `com.sun.identity.agents.config.response.header[Cache-Control]=no-cache`.

Default: not set

Property: `com.sun.identity.agents.config.response.header[`*`HEADER_NAME`*`]=`*`HEADER_VALUE`*

Redirect Attempt Limit

When set to a value other than zero, this defines the maximum number of redirects allowed for a single browser session, after which the Java agent blocks the request.

Default: `0`

Property: `com.sun.identity.agents.config.redirect.attempt.limit`

Agent Debug Level

Specifies the level of detail of the agent debug logs.

Valid values for the property are:

- `all`
- `error`

- `info`
- `message`
- `warning`

Set this property to `all` for fine-grain details.

Default: `error`

Property: `com.iplanet.services.debug.level`

Export Monitoring Metrics to CSV (Not yet in the AM console)[4]

When `true`, enables the export of Java agent performance monitoring metrics to comma-separated value (CSV) files.

The monitoring `.csv` files are written the same directory as the agent instance debug files, for example in `/path/to/java_agents/tomcat_agent/Agent_001/logs/debug/`.

Default: `false`

Property: `org.forgerock.agents.config.monitoring.to.csv`

User Mapping Properties

User Mapping Mode

Specifies the mechanism used to determine the user ID. This property can take four values:

- `USER_ID`. The Java agent reads the property `com.sun.identity.agents.config.user.principal`:
  - If `true`, the Java agent sets the principal user name as the user ID.
  - If `false`, the user ID is set to the value of the session property specified by the `com.sun.identity.agents.config.user.token` property as the user ID.
- `PROFILE_ATTRIBUTE`. The user ID is set to the value of a named profile attribute, as specified by the `com.sun.identity.agents.config.user.attribute.name` property.
- `HTTP_HEADER`. The user ID is set to the value of a named HTTP header, as specified by the `com.sun.identity.agents.config.user.attribute.name` property.
- `SESSION_PROPERTY`. The user ID is set to the value of a named session property, as specified by the `com.sun.identity.agents.config.user.attribute.name` property.

If the user ID cannot be set, the user will not be logged in and access requests will be denied.

Default: `USER_ID`

Property: `com.sun.identity.agents.config.user.mapping.mode`

User Attribute Name

Specifies the data store attribute that contains the user ID.

Default: `employeenumber`

Property: `com.sun.identity.agents.config.user.attribute.name`

User Principal Flag

When enabled, AM uses both the principal user name and also the user ID for authentication.

Default: `false`

Property: `com.sun.identity.agents.config.user.principal`

User Token Name

Specifies the session property name for the authenticated user's ID.

Default: `USER_ID`

Property: `com.sun.identity.agents.config.user.token`

Audit Properties

Audit Access Types

Specify the type of messages to log. Valid values include:

- `LOG_NONE`. Disable audit logging.
- `LOG_ALLOW`. Log access allowed events.
- `LOG_DENY`. Log access denied events.
- `LOG_BOTH`. Log access allowed and access denied events.

Default: `LOG_NONE`

Property: `com.sun.identity.agents.config.audit.accesstype`

Hot-swap: yes

Audit Log Location

Specifies the location where the Java agent audit message logs. Valid values include:

- `REMOTE`. Log audit event messages to the audit event handler configured in the AM realm.
- `LOCAL`. Log audit event messages locally to the agent installation.
- `ALL`. Log audit event messages to the audit event handler configured in the AM realm and locally to the agent installation.

Default: `REMOTE`

Property: `com.sun.identity.agents.config.log.disposition`

Hot-swap: yes

~~Remote Log File Name~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.agents.config.remote.logfile`

Rotate Local Audit Log

When enabled, rotate local audit log files that have reached the size specified by the Local Audit Log Rotation Size property.

Default: `false`

Property: `com.sun.identity.agents.config.local.log.rotate`

Hot-swap: yes

Local Audit Log Rotation Size

Specifies the maximum size in bytes of the local audit log files. When audit log rotation is enabled, the Java agent rotates the log file when it reaches this size.

Default: `52428800`

Property: `com.sun.identity.agents.config.local.log.size`

Hot-swap: yes

Fully Qualified Domain Name Checking Properties

FQDN Check

Enables checking of FQDN default value and FQDN map values.

Default: `false`

Property: `com.sun.identity.agents.config.fqdn.check.enable`

FQDN Default

FQDN users should use to access resources.

This property ensures that when users access protected resources on the web server without specifying the FQDN, the Java agent can redirect the users to URLs containing the correct FQDN.

> **Note**
>
> If you add any FQDN to this property, you must also add it to the Agent Root URL for CDSSO property.

Default: *agent-root-URL*

Property: `com.sun.identity.agents.config.fqdn.default`

FQDN Virtual Host Map

Maps virtual, invalid, or partial hostnames, and IP addresses to the FQDN to access protected resources. The property allows Java agents to redirect users to the FQDN and receive cookies belonging to the domain. It also ensures that invalid FQDN values that can cause the application server to become unusable or render resources inaccessible get properly mapped to the FQDN.

The property accepts an *invalid_hostname* and a *validN* Map Key value. The *invalid_hostname* maps an invalid or a partial hostname, or an IP address to a FQDN. The *validN* (where N = 1, 2, 3 ...) Map Key maps virtual hostnames to a FQDN.

```
com.sun.identity.agents.config.fqdn.mapping[invalid_hostname] = valid_hostname
com.sun.identity.agents.config.fqdn.mapping[validN] = valid_hostname
```

For example, to map the partial hostname `myserver` to `myserver.mydomain.example`, enter `myserver` in the Map Key field, enter `myserver.mydomain.example` in the Corresponding Map Value field and then click Add. This corresponds to:

```
com.sun.identity.agents.config.fqdn.mapping[myserver] = myserver.mydomain.example
```

To address a server as `xyz.hostname.com`, when the actual name of the server is `abc.hostname.com`, enter `valid1` in the Map Key field, enter `xyz.hostname.example` in the Corresponding Map Value field and then click Add. This corresponds to:

```
com.sun.identity.agents.config.fqdn.mapping[valid1] = xyz.hostname.com
```

If you have multiple virtual servers `rst.hostname.com`, `uvw.hostname.com`, and `xyz.hostname.com` pointing to the same actual server `abc.hostname.com` and each virtual server has its own policies defined, the properties can be defined as:

```
com.sun.identity.agents.config.fqdn.mapping[valid1] = rst.hostname.com
```

```
com.sun.identity.agents.config.fqdn.mapping[valid2] = uvw.hostname.com
com.sun.identity.agents.config.fqdn.mapping[valid3] = xyz.hostname.com
```

Default: not set

Property: `com.sun.identity.agents.config.fqdn.mapping`

Configuring Application Properties

After creating the agent profile, access the following properties in the AM console by navigating to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Application.

This section describes the following property groups:

- Login Processing Properties
- Logout Processing Properties
- Access Denied URI Processing Properties
- Not-Enforced Processing Properties
- Not-Enforced URI Processing Properties
- Not-Enforced IP Processing Properties
- Profile Attributes Processing Properties
- Response Attributes Processing Properties
- Common Attributes Fetching Processing Properties
- Session Attributes Processing Properties
- Privilege Attributes Processing Properties
- Custom Authentication Processing Properties
- Continuous Security Properties
- Query Parameter Handling Properties

Login Processing Properties

Login Form URI

Specifies the list of absolute URIs corresponding to a protected application's `web.xml form-login-page` element, such as `/myApp/jsp/login.jsp`.

Default: not set

Property: `com.sun.identity.agents.config.login.form`

Login Error URI

Specifies the list of absolute URIs corresponding to a protected application's `web.xml form-error-page` element, such as `/myApp/jsp/error.jsp`.

Default: not set

Property: `com.sun.identity.agents.config.login.error.uri`

Use Internal Login

When enabled, the Java agent uses the internal default content file for the login.

Default: `true`

Property: `com.sun.identity.agents.config.login.use.internal`

Login Content File Name

Full path name to the file containing custom login content when Use Internal Login is enabled.

Default: `FormLoginContent.txt`

Property: `com.sun.identity.agents.config.login.content.file`

Logout Processing Properties

~~Application Logout Handler~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.agents.config.logout.application.handler`

Application Logout URI

Specifies request URIs that indicate logout events. The key is the web application name. The value is the application logout URI.

To set a global logout URI for applications without other logout URIs defined, leave the key empty and set the value to the global logout URI, `/logout.jsp`.

To set a logout URI for a specific application, set the key to the name of the application, and the value to the application logout page.

Default: not set

Property: `com.sun.identity.agents.config.logout.uri`

Logout Request Parameter

Specifies parameters in the HTTP request that indicate logout events. The key is the web application name. The value is the logout request parameter.

To set a global logout request parameter for applications without other logout request parameters defined, leave the key empty and set the value to the global logout request parameter, `logoutparam`.

To set a logout request parameter for a specific application, set the key to the name of the application, and the value to the application logout request parameter, such as `logoutparam`.

Default: not set

Property: `com.sun.identity.agents.config.logout.request.param`

Logout Introspect Enabled

When enabled, the Java agent checks the HTTP request body to locate the Logout Request Parameter you set.

Default: `disabled`

Property: `com.sun.identity.agents.config.logout.introspect.enabled`

Logout Entry URI

Specifies the URIs to return after successful logout and subsequent authentication. The key is the web application name. The value is the URI to return.

To set a global logout entry URI for applications without other logout entry URIs defined, leave the key empty and set the value to the global logout entry URI, `/welcome.html`.

To set a logout entry URI for a specific application, set the key to the name of the application, and the value to the application logout entry URI, such as `/myApp/welcome.html`.

Default: not set

Property: `com.sun.identity.agents.config.logout.entry.uri`

Access Denied URI Processing Properties

Resource Access Denied URI

Specifies the URIs of custom pages to return when access is denied. The key is the web application name. The value is the custom URI.

To set a global custom access denied URI for applications without other custom access denied URIs defined, leave the key empty and set the value to the global custom access denied URI, `/sample/accessdenied.html`.

To set a custom access denied URI for a specific application, set the key to the name of the application, and the value to the application access denied URI, such as `/myApp/accessdenied.html`.

Default: not set

Property: `com.sun.identity.agents.config.access.denied.uri`

Not-Enforced Processing Properties

Not-Enforced Compound Rules Separator

Specifies a delimiter for the not-enforced compound rules. The delimiter can be a single character or a string. For example, setting the delimiter to `&&` allows compound rules to be specified as:

```
GET 10.5.1.5 100.2.21.36 && /public/*
REGEX 10\.4\.3\.5 && [^/]+\/free.jpg
```

Default: `|`

Property:`org.forgerock.openam.agents.config.notenforced.rule.compound.separator`

Not-Enforced URI Processing Properties

For more information about not-enforced rule evaluation and caching, see "Not-Enforced Lists".

Not-Enforced URIs

Specifies a space-delimited list of URIs for which no authentication is required. For example:

```
/public/* /images/*
```

If you are using a local configuration file rather than the administration console, use the following format when specifying not-enforced URIs, where *n* is a unique, incrementing integer:

```
com.sun.identity.agents.config.notenforced.uri[n]=Not enforced URI Rule
```

If the URI contains spaces or other reserved characters, you must percent-encode (often referred to as URL encoding) them. For example, `/my%20public%20app/`.

To fine-tune the not-enforced URI list, the Java agent supports inverting rules, and using regular expressions and wildcards. You can also filter based on HTTP methods, and cookie and header values:

- **Inverting Rules**

Not-enforced URI rules can be inverted either by rule or by property:

- **By rule**. Invert any rule in the Not-Enforced URIs property by preceding it with the keyword `NOT`, separated by a space (blank) character. In the following example, the agent will defer to AM any request of a `.jpg` file in the `/private` URI:

  ```
  NOT /private/*.jpg
  ```

- **By property**. Invert all the rules in the Not-Enforced URIs property by setting the Invert Not-Enforced URIs property to `true`.

- **Wildcards, Regular Expressions, HTTP Methods, Cookie and Header Values**

  For finer control over the filtering of not-enforced URI rules, you can create rules that use wildcards or regular expressions, and that filter HTTP methods, cookie values, and headers.

  - **Wildcards**

    Not-Enforced URI rules support two types of wildcards:

    - The * wildcard matches all characters except the question mark **?** character. It cannot be escaped, and spans multiple levels in a URI. For example:

      ```
      /images/*
      /*.jsp?locale=*
      ```

      Multiple forward slashes do not match a single forward slash, so * matches `mult/iple/dirs`, yet `mult/*/dirs` does not match `mult/dirs`.

    - The **-*-** wildcard matches all characters except the forward slash **/** and the question mark **?** character. It cannot be escaped. Because it does not match the **/** character, the **-*-** wildcard does not span multiple levels in a URI. For example:

      ```
      /css/-*-
      ```

    When using wildcards on not-enforced URI rules, consider the following points:

    - The use of the * and **-*-** wildcards in the same rule is not supported. However, you can use them in different rules in the same list. For example:

      ```
      /css/-*-
      /images/*
      ```

    - Multiple wildcards in the query parameter section of a not-enforced URI rule will match the parameters in any order that they might appear in a resource URI. For example:

      ```
      /customers/*?*member_level=*location=*
      ```

      The not-enforced URI rule above will apply to any resource URI that contains a `member_level` and `location` query parameter, in any order. In this example, the following requests would be not-enforced:

```
https://www.example.com/customers/default.jsp?member_level=silver&location=fr
https://www.example.com/customers/default.jsp?location=es&member_level=silver
https://www.example.com/customers/default.jsp?location=uk&vip=true&member_level=gold
```

If the parameters are not present, the agent falls back to evaluating the resource URI against policies in AM, as usual.

- Trailing forward slashes are not recognized as part of a resource name. Therefore, `/images//` and `/images` are equivalent.

For more information on wildcard usage, see Specifying Resource Patterns with Wildcards.

- **Regular Expressions**

  To use regular expressions in a not-enforced URI rule, add the keyword `REGEX` followed by a blank (space) character before the URI to match. For example:

  ```
  REGEX https?://www\.example\.com/([^/])+/.*\.jpg
  ```

  When using regular expressions in a not-enforced URI list, consider the following points:

  - The use of 'classic' wildcards and regular expressions in the same rule is not supported.
  - If an invalid regular expression is specified in a rule, the rule is dropped and an error message will show in the logs.

- **HTTP Methods**

  Specify not-enforced HTTP methods by adding one of the following keywords: `GET`, `HEAD`, `POST`, `PUT`, `PATCH`, `DELETE`, `OPTIONS`, and `TRACE`.

  By default, if no HTTP method is specified for a particular rule, all methods are not-enforced for that rule. For example, the following rule does not enforce every supported HTTP method for any file contained in `/public`:

  ```
  /public/*
  ```

  To specify which methods should not be enforced, add a comma-delimited list of methods followed by a blank (space) character before the URL to match. For example:

  ```
  GET,POST /public/*
  GET,POST,PUT /examples/notenforced/*.jpg
  GET,REGEX https?://www\.example\.com/([^/])+/.*\.jpg
  ```

  Any method that is not specified will be enforced.

  Methods can be inverted by adding an exclamation mark **!** character in front of them. For example, all methods but `POST` are not enforced in the following example:

  ```
  !POST /public/*
  ```

Unrecognized keywords in a rule are ignored and do not invalidate the rest of the rule.

- **Cookie Values**

  You can create not-enforced rules that only apply when the incoming request has a named cookie, with a specified value.

  The syntax for specifying rules that apply to cookies with a specified value is as follows:

  ```
  COOKIE(Name/Value/Modifiers) Not Enforced URIs
  ```

  Where:

  Name

  Specifies the name of the cookie to inspect for the specified value.

  The name of the cookie is case-sensitive.

  Value

  Specifies a string to look for in the value field of the specified cookie.

  Modifiers

  Specify one or more of the following optional modifiers to alter the method for looking up the value:

  i

  Perform a case-insensitive comparison.

  r

  Treat the string specified in *Value* as a regular expression.

  For example, to allow access to all resources in `/private/admin/images/` when there is a cookie named `login_result` (case-sensitive) present on the request that has a value `VALID`, ignoring case, specify a rule similar to the following:

  ```
  COOKIE(login_result/VALID/i) /private/admin/images/*
  ```

  You can combine cookie filters with other filters, such as HTTP methods.

  For example, the following rule allows `GET`, `POST`, and `PUT` HTTP requests to HTML resources in the `/other/records/` folder, providing that there is a cookie named `internal` (case-sensitive) present, and it has a value that matches the regular expression `.*ID` - that is; strings that end with `ID` - ignoring case:

  ```
  GET,POST,COOKIE(internal/.*ID/ri),PUT /other/records/*.html
  ```

Note that combining a `HEADER` and `COOKIE` expression in the same rule implies a logical AND is applied, such that both expressions must match in order to apply. To apply the rules as a logical OR, create two separate rules.

- **Header Values**

  You can create not-enforced rules that only apply when the incoming request has a named header, with a specified value.

  The syntax for specifying rules that apply to headers with a specified value is as follows:

  ```
  HEADER(Name/Value/Modifiers) Not Enforced URIs
  ```

  Where:

  `Name`

  Specifies the name of the header to inspect for the specified value.

  The name of the header is *not* case-sensitive.

  `Value`

  Specifies a string to look for in the value field of the specified header.

  `Modifiers`

  Specify one or more of the following optional modifiers to alter the method for looking up the value:

  `i`

  Perform a case-insensitive string comparison.

  `r`

  Treat the string specified in ***`Value`*** as a regular expression.

  For example, to allow access to all TXT files in `/yearly/2019/` when there is a header named `ID` present on the request that has a value `validated`, ignoring case, specify a rule similar to the following:

  ```
  HEADER(ID/validated/i) /yearly/2019/*.txt
  ```

  You can combine cookie filters with other filters, such as HTTP methods.

  For example, the following rule allows `GET`, `POST`, and `PUT` HTTP requests to HTML resources in the `/other/records/` folder, providing that there is a header named `internal` present, and it has a value that matches the regular expression `.*ID` - that is; strings that end with `ID` - ignoring case:

```
GET,POST,HEADER(internal/.*ID/ri),PUT /other/records/*.html
```

Note that combining a `HEADER` and `COOKIE` expression in the same rule implies a logical AND is applied, such that both expressions must match in order to apply. To apply the rules as a logical OR, create two separate rules.

- **Compound Not-Enforced Rules**

  Compound not-enforced rules allow you to combine not-enforced URI and IP rules in a single rule. They can be configured either in the Not-Enforced URIs or the Not-Enforced Client IP List properties.

  To write not-enforced URI and IP rules, follow the guidelines explained in Not-Enforced IP Processing Properties and Not-Enforced URI Processing Properties.

  The format for compound rules requires the IP rule or list of IP rules, a delimiter, by default the horizontal line **|** character, and the URI rule or list of URI rules. Blank (space) characters around the delimiter are optional. For example:

  ```
  192.168.1.1-192.168.4.3 | /images/*
  ```

  In the example, the agent will not enforce any HTTP requests coming from the ip range `192.168.1.1-192.168.4.3` to any file (*) in the `/images` URI.

  When configuring compound rules, consider the following points:

  - Keywords, such as HTTP methods, `NOT`, and `REGEX`, are required at the beginning of the compound rule, and affect both the IP and the URI rules. For example:

    ```
    GET,POST 192.168.1.1-192.168.4.3 | /images/*
    ```

    In the preceding example, the agent will not enforce `GET` and `POST` HTTP requests coming from the ip range `192.168.1.1-192.168.4.3` to any file (*) in the `/images` URI.

    ```
    NOT,!POST 192.168.1.* | /private/*
    ```

    In the preceding example, the agent will defer to AM (`NOT` keyword) any request done to all supported HTTP methods but `POST` (**!**) coming from any ip address in the `192.168.1` subnet to any file (*) in the `/private` URI.

  - When working with the `REGEX` keyword, ensure both sides of the rule can be parsed as a regular expression. For example:

    ```
    POST,REGEX 192\.168\.10\.(10|\d) && \/images\/([^/])+\.*\.jpg
    ```

    Note that the delimiter in the previous example is **&&**. This is because the **|** character can lead to invalid regular expressions. To configure a different delimiter, see the Not-Enforced Compound Rules Multi-Value Separator property in Not-Enforced Processing Properties.

- When dealing with compound rules, the agent caches hits and misses for each resource accessed. IP and URI not-enforced lists have a property each to enable or disable their caches; for compound rules, caching is enabled if either the IP or URI not-enforced cache is enabled. The size of its cache has the size of the larger of the two IP or URI cache sizes.

  For more information about not-enforced rule evaluation and caching, see "Not-Enforced Lists".

- **Encoding Internationalized Resource Identifiers (IRIs)**

  To match a resource that uses non-ASCII characters, percent-encode the resource when creating the rule.

  For example, to match resources under an IRI such as `http://www.example.com/forstå`, specify the following percent-encoded rule:

  ```
  /forst%C3%A5/*
  ```

Default: not set

Property: `com.sun.identity.agents.config.notenforced.uri[n]`

Invert Not-Enforced URIs

When set to `true`, enforce policy for the URIs and patterns specified in the Not-Enforced URIs property instead of allowing access to them without authentication.

> **Note**
>
> ForgeRock recommends using the `NOT` keyword to invert specific rules in the Not-Enforced URI list, instead of inverting all rules by setting the Invert Not-Enforced URIs property to `true`.

Default: `false`

Property: `com.sun.identity.agents.config.notenforced.uri.invert`

Not-Enforced URIs Cache Enabled

When set to `true`, the agent caches evaluation (hits and misses) of the not-enforced URI list. When configuring many rules (in the hundreds) this setting should be enabled.

For more information about not-enforced caching, see "Not-Enforced Lists".

Default: `true`

Property: `com.sun.identity.agents.config.notenforced.uri.cache.enable`

Not-Enforced URIs Cache Size

When caching is enabled, this limits the number of not-enforced URIs cached.

Default: `10000`

Property: `com.sun.identity.agents.config.notenforced.uri.cache.size`

Refresh Session Idle Time

When set to `true`, the agent resets the CTS-based session idle time when granting access to a not-enforced URI, prolonging the time before the user must authenticate again. This setting has no effect on users with client-based (stateless) sessions.

Default: `false`

Property: `com.sun.identity.agents.config.notenforced.refresh.session.idletime`

Not-Enforced IP Processing Properties

For more information about not-enforced rule evaluation and caching, see "Not-Enforced Lists".

Not-Enforced Client List

Specifies a space-delimited list of IP addresses or network CIDR notation for which no authentication is required.

Supported values are IPV4 and IPV6 addresses, IPV4 and IPV6 ranges of addresses delimited by the **-** character, and network ranges specified in CIDR notation. For example:

```
192.168.1.0/24 192.168.100.0/24
2001:5c0:9168:0:0:0:0:2/128
192.168.1.1-192.168.4.3
2001:5c0:9168:0:0:0:0:1-2001:5c0:9168:0:0:0:0:2
```

If you are using a local configuration file rather than the administration console, use the following format when specifying not-enforced client lists, where *n* is a unique, incrementing integer:

```
com.sun.identity.agents.config.notenforced.ip[n]=Not enforced IP Rule
```

To fine-tune the not-enforced IP list, the Java agent supports inverting rules, using regular expressions and wildcards, and specifying HTTP methods:

- **Inverting Rules**

  Not-enforced IP rules can be inverted either by rule or by property:

  - **By rule.** Invert any rule in the Not-Enforced Client IP List property by preceding it with the keyword `NOT`, separated by a space (blank) character. In the following example, the agent will defer to AM any request from the network specified by the `192.168.1.0/24` CIDR notation:

    ```
    NOT 192.168.1.0/24
    ```

  - **By property**. Invert all the rules in the Not-Enforced Client IP List property by setting the Not-Enforced IP Invert List to `true`.

- **Wildcards, Regular Expressions, and HTTP Methods**

For finer control over the filtering of not-enforced IP rules, use wildcards, regular expressions, HTTP methods, cookie values, and headers:

- **Wildcards**

  The * wildcard matches all characters except the question mark **?** character, and cannot be escaped. For example:

  ```
  192.168.*
  ```

  For more information on wildcard usage, see Specifying Resource Patterns with Wildcards.

- **Regular Expressions**

  To use regular expressions in a not-enforced IP rule, add the keyword `REGEX` followed by a blank (space) character before the URI to match. For example:

  ```
  REGEX 192\.168\.10\.\d+
  ```

  When using regular expressions in a not-enforced IP list, consider the following points:

  - The use of wildcards and regular expressions in the same rule is not supported.
  - The use of netmask CIDR notation or ip address ranges and regular expressions is not supported. However, you can create a regular expression that matches a range of IP addresses, such as:

    ```
    REGEX 192\.168\.10\.(10|\d)
    ```

  - If an invalid regular expression is specified in a rule, the rule is dropped and an error message will show in the logs.

- **HTTP Methods**

  Specify not-enforced HTTP methods by adding one of the following keywords: `GET`,`HEAD`, `POST`, `PUT`, `PATCH`, `DELETE`, `OPTIONS`, and `TRACE`.

  By default, if no HTTP method is specified for a particular rule, all methods are not-enforced for that rule. For example, the following rule does not enforce every supported HTTP method for the ips specified by `192.168.10.*`:

  ```
  192.168.10.*
  ```

  To specify which methods should not be not-enforced, add a comma-delimited list of methods followed by a blank (space) character before the URL to match. For example:

  ```
  NOT,GET,REGEX 192\.168\.10\.\d+
  POST 192.168.10.*
  GET 192.168.10.1-192.168.10.254 192.168.0.1
  POST,PUT 192.168.1.0/24
  ```

Any method that is not specified will be enforced.

Methods can be inverted by adding an exclamation point **!** character in front of them. For example, all methods but `POST` are enforced in the following example:

```
!POST 192.168.1.0/24
```

Unrecognized keywords in a rule are ignored and do not invalidate the rest of the rule.

- **Cookie Values**

You can create not-enforced rules that only apply when the incoming request has a named cookie, with a specified value.

The syntax for specifying rules that apply to cookies with a specified value is as follows:

```
COOKIE(Name/Value/Modifiers) Not Enforced IPs
```

Where:

Name

Specifies the name of the cookie to inspect for the specified value.

The name of the cookie is case-sensitive.

Value

Specifies a string to look for in the value field of the specified cookie.

Modifiers

Specify one or more of the following optional modifiers to alter the method for looking up the value:

i

Perform a case-insensitive comparison.

r

Treat the string specified in *Value* as a regular expression.

For example, to allow access from `192.168.*` when there is a cookie named `login_result` (case-sensitive) present on the request that has a value `VALID`, ignoring case, specify a rule similar to the following:

```
COOKIE(login_result/VALID/i) 192.168.*
```

You can combine cookie filters with other filters, such as HTTP methods.

For example, the following rule allows `GET`, `POST`, and `PUT` HTTP requests from the client IP range `192.168.*`, providing that there is a cookie named `internal` (case-sensitive) present, and it has a value that matches the regular expression `.*ID` - that is; strings that end with `ID` - ignoring case:

```
GET,POST,COOKIE(internal/.*ID/ri),PUT 192.168.*
```

Note that combining a `HEADER` and `COOKIE` expression in the same rule implies a logical AND is applied, such that both expressions must match in order to apply. To apply the rules as a logical OR, create two separate rules.

- **Header Values**

  You can create not-enforced rules that only apply when the incoming request has a named header, with a specified value.

  The syntax for specifying rules that apply to headers with a specified value is as follows:

  ```
  HEADER(Name/Value/Modifiers) Not Enforced IPs
  ```

  Where:

  Name

  Specifies the name of the header to inspect for the specified value.

  The name of the header is *not* case-sensitive.

  Value

  Specifies a string to look for in the value field of the specified header.

  Modifiers

  Specify one or more of the following optional modifiers to alter the method for looking up the value:

  i

  Perform a case-insensitive string comparison.

  r

  Treat the string specified in *Value* as a regular expression.

For example, to allow access to the IP range `192.168.*` when there is a header named `ID` present on the request that has a value `validated`, ignoring case, specify a rule similar to the following:

```
HEADER(ID/validated/i) 192.168.*
```

You can combine cookie filters with other filters, such as HTTP methods.

For example, the following rule allows `GET`, `POST`, and `PUT` HTTP requests from the IP address range `192.168.*`, providing that there is a header named `internal` present, and it has a value that matches the regular expression `.*ID` - that is; strings that end with `ID` - ignoring case:

```
GET,POST,HEADER(internal/.*ID/ri),PUT 192.168.*
```

Note that combining a `HEADER` and `COOKIE` expression in the same rule implies a logical AND is applied, such that both expressions must match in order to apply. To apply the rules as a logical OR, create two separate rules.

- **Compound Not-Enforced Rules**

  Compound not-enforced rules allow you to combine not-enforced URI and IP rules in a single rule. They can be configured either in the Not-Enforced URIs or the Not-Enforced Client IP List properties.

  To write not-enforced URI and IP rules, follow the guidelines explained in Not-Enforced IP Processing Properties and Not-Enforced URI Processing Properties.

  The format for compound rules requires the IP rule or list of IP rules, a delimiter, by default the horizontal line **|** character, and the URI rule or list of URI rules. Blank (space) characters around the delimiter are optional. For example:

  ```
  192.168.1.1-192.168.4.3 | /images/*
  ```

  In the example, the agent will not enforce any HTTP requests coming from the ip range `192.168.1.1-192.168.4.3` to any file (*) in the `/images` URI.

  When configuring compound rules, consider the following points:

  - Keywords, such as HTTP methods, `NOT`, and `REGEX`, are required at the beginning of the compound rule, and affect both the IP and the URI rules. For example:

    ```
    GET,POST 192.168.1.1-192.168.4.3 | /images/*
    ```

    In the preceding example, the agent will not enforce `GET` and `POST` HTTP requests coming from the ip range `192.168.1.1-192.168.4.3` to any file (*) in the `/images` URI.

    ```
    NOT,!POST 192.168.1.* | /private/*
    ```

    In the preceding example, the agent will defer to AM (`NOT` keyword) any request done to all supported HTTP methods but `POST` (**!**) coming from any ip address in the `192.168.1` subnet to any file (*) in the `/private` URI.

  - When working with the `REGEX` keyword, ensure both sides of the rule can be parsed as a regular expression. For example:

    ```
    POST,REGEX 192\.168\.10\.(10|\d) && \/images\/([^/])+\.*\.jpg
    ```

Note that the delimiter in the previous example is **&&**. This is because the **|** character can lead to invalid regular expressions. To configure a different delimiter, see the Not-Enforced Compound Rules Multi-Value Separator property in Not-Enforced Processing Properties.

- When dealing with compound rules, the agent caches hits and misses for each resource accessed. IP and URI not-enforced lists have a property each to enable or disable their caches; for compound rules, caching is enabled if either the IP or URI not-enforced cache is enabled. The size of its cache has the size of the larger of the two IP or URI cache sizes.

  For more information about not-enforced rule evaluation and caching, see "Not-Enforced Lists".

- **Encoding Internationalized Resource Identifiers (IRIs)**

  To match a resource that uses non-ASCII characters, percent-encode the resource when creating the rule.

  For example, to match resources under an IRI such as `http://www.example.com/forstå`, specify the following percent-encoded rule:

  ```
  /forst%C3%A5/*
  ```

Default: not set

Property: `com.sun.identity.agents.config.notenforced.ip[n]`

Not-Enforced IP Invert List

When set to `true`, enforce policy for the IPs specified in the Not-Enforced Client IP List property instead of allowing access to them without authentication.

> **Note**
>
> ForgeRock recommends using the `NOT` keyword to invert specific rules in the Not-Enforced Client IP List, instead of inverting all the rules by setting the Not-Enforced IP Invert List property to `true`.

Default: `false`

Property: `com.sun.identity.agents.config.notenforced.ip.invert`

Not-Enforced IP Cache Flag

When set to `true`, the agent caches evaluation (hits and misses) of the not-enforced IP list. When configuring many rules (in the hundreds), this setting must be enabled.

For more information about not-enforced caching, see "Not-Enforced Lists".

Default:`true`

Property: `com.sun.identity.agents.config.notenforced.ip.cache.enable`

Not-Enforced IP Cache Size

When caching is enabled, this limits the number of not-enforced addresses cached.

Default: `10000`

Property: `com.sun.identity.agents.config.notenforced.ip.cache.size`

Profile Attributes Processing Properties

Profile Attribute Fetch Mode

When set to `HTTP_COOKIE` or `HTTP_HEADER`, profile attributes are introduced into the cookie or the headers, respectively. When set to `REQUEST_ATTRIBUTE`, profile attributes are part of the HTTP request.

Property: `com.sun.identity.agents.config.profile.attribute.fetch.mode`

Profile Attribute Mapping

Maps the profile attributes to HTTP headers for the currently authenticated user. Map Keys are attribute names, and Map Values are HTTP header names. The user profile can be stored in LDAP or any other arbitrary data store.

To populate the value of profile attribute CN under `CUSTOM-Common-Name`: enter CN in the Map Key field, and enter `CUSTOM-Common-Name` in the Corresponding Map Value field. This corresponds to `com.sun.identity.agents.config.profile.attribute.mapping[cn]=CUSTOM-Common-Name`.

In most cases, in a destination application where an HTTP header name shows up as a request header, it is prefixed by `HTTP_`, lower case letters become upper case, and hyphens (`-`) become underscores (`_`). For example, `common-name` becomes `HTTP_COMMON_NAME`.

Property: `com.sun.identity.agents.config.profile.attribute.mapping`

Response Attributes Processing Properties

Response Attribute Fetch Mode

When set to `HTTP_COOKIE` or `HTTP_HEADER`, response attributes are introduced into the cookie or the headers, respectively. When set to `REQUEST_ATTRIBUTE`, response attributes are part of the HTTP request.

Property: `com.sun.identity.agents.config.response.attribute.fetch.mode`

Response Attribute Mapping

Maps the policy response attributes to HTTP headers for the currently authenticated user. The response attribute is the attribute in the policy response to be fetched.

To populate the value of response attribute `uid` under `CUSTOM-User-Name`: enter `uid` in the Map Key field, and enter `CUSTOM-User-Name` in the Corresponding Map Value field. This corresponds to `com.sun.identity.agents.config.response.attribute.mapping[uid]=Custom-User-Name`.

In most cases, in a destination application where an HTTP header name shows up as a request header, it is prefixed by `HTTP_`, lower case letters become upper case, and hyphens (`-`) become underscores (`_`). For example, `response-attr-one` becomes `HTTP_RESPONSE_ATTR_ONE`.

Property: `com.sun.identity.agents.config.response.attribute.mapping`

Common Attributes Fetching Processing Properties

Cookie Separator Character

Specifies the separator for multiple values of the same attribute when it is set as a cookie.

Default: `|`

Property: `com.sun.identity.agents.config.attribute.cookie.separator`

Fetch Attribute Date Format

Specifies the `java.text.SimpleDateFormat` of date attribute values used when an attribute is set in an HTTP header.

Default: `EEE, d MMM yyyy hh:mm:ss z`.

Property: `com.sun.identity.agents.config.attribute.date.format`

Attribute Cookie Encode

When enabled, attribute values are URL-encoded before being set as a cookie.

Default: `true`

Property: `com.sun.identity.agents.config.attribute.cookie.encode`

Session Attributes Processing Properties

Session Attribute Fetch Mode

When set to `HTTP_COOKIE` or `HTTP_HEADER`, session attributes are introduced into the cookie or the headers, respectively. When set to `REQUEST_ATTRIBUTE`, session attributes are part of the HTTP request.

Property: `com.sun.identity.agents.config.session.attribute.fetch.mode`

Session Attribute Mapping

Maps session attributes to HTTP headers for the currently authenticated user. The session attribute is the attribute in the session to be fetched.

To populate the value of session attribute `UserToken` under `CUSTOM-userid`: enter `UserToken` in the Map Key field, and enter `CUSTOM-userid` in the Corresponding Map Value field. This corresponds to `com.sun.identity.agents.config.session.attribute.mapping[UserToken]=CUSTOM-userid`.

In most cases, in a destination application where an HTTP header name shows up as a request header, it is prefixed by `HTTP_`, lower case letters become upper case, and hyphens (`-`) become underscores (`_`). For example, `success-url` becomes `HTTP_SUCCESS_URL`.

Property: `com.sun.identity.agents.config.session.attribute.mapping`

Privilege Attributes Processing Properties

The following properties do not apply to Java Agents 5.6, although they may appear in the AM console:

~~Default Privileged Attribute~~

Property: `com.sun.identity.agents.config.default.privileged.attribute`

~~Privileged Attribute Type~~

Property: `com.sun.identity.agents.config.privileged.attribute.type`

~~Privileged Attributes To Lower Case~~

Property: `com.sun.identity.agents.config.privileged.attribute.tolowercase`

~~Privileged Session Attribute~~

Property: `com.sun.identity.agents.config.privileged.session.attribute`

~~Enable Privileged Attribute Mapping~~

Property: `com.sun.identity.agents.config.privileged.attribute.mapping.enable`

~~Privileged Attribute Mapping~~

Property: `com.sun.identity.agents.config.privileged.attribute.mapping`

Custom Authentication Processing Properties

The following properties do not apply to Java Agents 5.6, although they may appear in the AM console:

~~Custom Authentication Handler~~

Property: `com.sun.identity.agents.config.auth.handler`

~~Custom Logout Handler~~

Property: `com.sun.identity.agents.config.logout.handler`

~~Custom Verification Handler~~

Property: `com.sun.identity.agents.config.verification.handler`

Continuous Security Properties

For more information about continuous security, see "Continuous Security".

Continuous Security Cookies

Maps cookie values available in inbound resource requests to entries in the environmental conditions map, which Java agents send to AM during policy evaluation.

This property has the format `[cookie_name]=map_entry_name`, where:

- `[cookie_name]` specifies the name of the cookie in the inbound request.
- `map_entry_name` specifies the name of the entry within the environmental conditions map that contains the value of `cookie_name`.

Example:

```
org.forgerock.openam.agents.config.continuous.security.cookies[trackingcookie1]=myCookieEntry
```

Java agents add entries from both of the continuous security properties into the environmental conditions map, which AM's authorization framework accesses during policy evaluation.

Use server-side authorization scripts to:

- Access the map's contents
- Write scripted conditions based on cookies and headers in the request

For more information about server-side authorization scripts in AM, see the *ForgeRock Access Management Authorization Guide*.

When you specify continuous security properties, Java agents generate environmental condition entries in the map as follows:

Key	Value
`requestIp` [a]	Contains the inbound request's IP address. The Java agent determines the IP as follows: • If the `com.sun.identity.agents.config.client.ip.header` property is configured, the agent extracts the IP address from the header.

Key	Value
	• If the `com.sun.identity.agents.config.client.ip.header` property is not configured, the agent uses the `HttpServletRequest.getRemoteAddr` Java function to determine the IP address.
`requestDNSName` [b]	Contains the inbound request's host name. The Java agent determines the host name as follows: • If the `com.sun.identity.agents.config.client.hostname.header` property is configured, the agent extracts the host name from the header. • If the `com.sun.identity.agents.config.client.hostname.header` property is not configured, the agent uses the `HttpServletRequest.getRemoteHost` Java function to determine the host name.
`variable_name` [c]	Contains an array of cookie or header values.

[a] The `requestIp` entry is created in the map regardless of how the continuous security properties are configured.
[b] The `requestDNSName` entry is created in the map regardless of how the continuous security properties are configured.
[c] There may be as many *`variable_name`* entries as values specified in the continuous security properties.

Consider the following example:

```
org.forgerock.openam.agents.config.continuous.security.cookies[ssid]=mySsid
org.forgerock.openam.agents.config.continuous.security.headers[User-Agent]=myUser-Agent
```

Assuming the incoming request contains an `ssid` cookie and an `User-Agent` header, the environmental conditions map would contain the following variables:

- `requestIp`, containing the IP address of the client. For example, `192.16.8.0.1`.
- `requestDNSName`, containing the host name of the client. For example, `client.example.com`.
- `mySsid`, containing the value of the `ssid` cookie. For example, `77xe99f4zqi1l99z`.
- `myUser-Agent`, containing the value of the `from` header. For example, `Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko`.

Default: not set

Property: `org.forgerock.openam.agents.config.continuous.security.cookies[`*`cookie_name`*`]=`*`map_entry_name`*

Continuous Security Headers

Maps header values in inbound resource requests to entries in the environmental conditions map, which Java agents send to AM during policy evaluation.

This property has the format `[`*`header_name`*`]=`*`map_entry_name`*, where:

- `[`*`header_name`*`]` specifies the name of the header in the inbound request.
- *`map_entry_name`* specifies the name of the entry within the environmental conditions map that contains the value of *`header_name`*.

Example:

```
org.forgerock.openam.agents.config.continuous.security.headers[User-Agent]=myUserAgentHeaderEntry
```

Java agents add entries from both of the continuous security properties into the environmental conditions map, which AM's authorization framework accesses during policy evaluation.

Use server-side authorization scripts to:

- Access the map's contents
- Write scripted conditions based on cookies and headers in the request

For more information about server-side authorization scripts in AM, see the *ForgeRock Access Management Authorization Guide*.

When you specify continuous security properties, Java agents generate environmental condition entries in the map as follows:

Key	Value
`requestIp` [a]	Contains the inbound request's IP address. The Java agent determines the IP as follows: • If the `com.sun.identity.agents.config.client.ip.header` property is configured, the agent extracts the IP address from the header. • If the `com.sun.identity.agents.config.client.ip.header` property is not configured, the agent uses the `HttpServletRequest.getRemoteAddr` Java function to determine the IP address.
`requestDNSName` [b]	Contains the inbound request's host name. The Java agent determines the host name as follows: • If the `com.sun.identity.agents.config.client.hostname.header` property is configured, the agent extracts the host name from the header. • If the `com.sun.identity.agents.config.client.hostname.header` property is not configured, the agent uses the `HttpServletRequest.getRemoteHost` Java function to determine the host name.
`variable_name` [c]	Contains an array of cookie or header values.

[a] The `requestIp` entry is created in the map regardless of how the continuous security properties are configured.
[b] The `requestDNSName` entry is created in the map regardless of how the continuous security properties are configured.
[c] There may be as many *`variable_name`* entries as values specified in the continuous security properties.

Consider the following example:

```
org.forgerock.openam.agents.config.continuous.security.cookies[ssid]=mySsid
org.forgerock.openam.agents.config.continuous.security.headers[User-Agent]=myUser-Agent
```

Assuming the incoming request contains an `ssid` cookie and an `User-Agent` header, the environmental conditions map would contain the following variables:

- `requestIp`, containing the IP address of the client. For example, `192.16.8.0.1`.
- `requestDNSName`, containing the host name of the client. For example, `client.example.com`.
- `mySsid`, containing the value of the `ssid` cookie. For example, `77xe99f4zqi1l99z`.
- `myUser-Agent`, containing the value of the `from` header. For example, `Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko`.

Default: not set

Property: `org.forgerock.openam.agents.config.continuous.security.headers[header_name]=map_entry_name`

Query Parameter Handling Properties

For more information about how the Java agent handles query parameters, see "Query Parameter Handling".

Remove Query Parameters (Not yet in the AM console)[4]

Specifies a list of query parameters to be removed from a URL for policy decision and caching purposes. The property has the format `[Domain/path] | parameter[,parameter...]` with no spaces between values. Specify values as follows:

- `Domain/path`

  Specifies the incoming request URL. It can take the following values:

  - A domain. For example, `example.com`.

    When you specify a domain, Java agents match both the domain itself and its subdomains. For example, `example.com` matches `mydomain.example.com` and `www.example.com`.

    Domains can also include path information, for example, `example.com/market`, but cannot specify ports.

  - A subdomain. For example, `mydomain.example.com`.

    When you specify a subdomain, Java agents match the domain, the subdomain, and any sub-subdomain. For example, `mydomain.example.com` matches `true.mydomain.example.com`, too.

    Subdomains can include path information, for example, `mydomain.example.com/secure`, but cannot specify ports.

  - A path. For example, `/myapp`.
  - No value, in which case nothing is specified before the | character and the rule applies to every incoming request.

Note

It is possible to specify both the Request-Header and Domain/path login formats in an incoming request via the agent properties. If both formats are specified, the Request-Header entries are applied first. If none of the headers in the incoming request match, the Domain/path entries will be applied.

- `parameter[,parameter...]`

  Specifies a comma-separated list of query parameters to remove from the incoming request URL.

  Consider the following constraints when constructing the list of parameters:

  - Add a comma (,) character at the beginning or the end of the list to remove all unnamed parameters. For example, `myapp.example.com/customers|,lang` would match both `lang` and any unnamed parameters.
  - Add the asterisk (*) character to the list to remove all parameters, including unnamed ones.

  The remaining parameters (those that do not match the list of parameters) are sorted alphabetically.

Examples:

```
org.forgerock.openam.agents.config.conditional.unwanted.http.url.params[0]=myapp.example.com/
customers|location,lang
org.forgerock.openam.agents.config.conditional.unwanted.http.url.params[1]=example.com/customers|*
```

For example, an incoming URL request such as `http://myapp.example.com/customers?country=uk&=bristol&lang=en_GB&area=1343456` that matches a rule such as `myapp.example.com/customers|,lang`, will be cached by the agent as `http://myapp.example.com/customers?area=1343456&country=uk`, where both `lang` and the unnamed parameter are removed and the rest of the parameters are sorted alphabetically.

Property: `org.forgerock.openam.agents.config.conditional.unwanted.http.url.params[n]`

Hot-swap: Yes

Regular Expression Remove Query Parameters (Not yet in the AM console)[4]

Specifies a list of regular expressions the agent uses to match query parameters to be removed from a URL for policy decision and caching purposes. The property has the format `[Domain/path] | regular_expression[,regular_expression...]` with no spaces between values. Specify values as follows:

- `Domain/path`

  Specifies the incoming request URL. It can take the following values:

  - A domain. For example, `example.com`.

When you specify a domain, Java agents match both the domain itself and its subdomains. For example, `example.com` matches `mydomain.example.com` and `www.example.com`.

Domains can also include path information, for example, `example.com/market`, but cannot specify ports.

- A subdomain. For example, `mydomain.example.com`.

  When you specify a subdomain, Java agents match the domain, the subdomain, and any sub-subdomain. For example, `mydomain.example.com` matches `true.mydomain.example.com`, too.

  Subdomains can include path information, for example, `mydomain.example.com/secure`, but cannot specify ports.

- A path. For example, `/myapp`.
- No value, in which case nothing is specified before the | character and the rule applies to every incoming request.

Note

It is possible to specify both the Request-Header and Domain/path login formats in an incoming request via the agent properties. If both formats are specified, the Request-Header entries are applied first. If none of the headers in the incoming request match, the Domain/path entries will be applied.

- `regular_expression[,regular_expression...]`

  Specifies a comma-separated list of regular expressions the agent uses to match query parameters to be removed from the incoming request URL.

  Consider the following constraints when constructing your list of regular expressions:

  - Add a comma (,) character at the beginning or the end of the list to remove all unnamed parameters. For example, `myapp.example.com/customers|,lang` would match both `lang` and any unnamed parameters.
  - Consider creating multiple simple regular expressions instead of a single complicated one.

  The remaining parameters (those that do not match the list of parameters) are sorted alphabetically.

Examples:

```
org.forgerock.openam.agents.config.conditional.unwanted.http.url.params.regexp[0]=myapp.example.com|
b.*,gp(a|p|s),
org.forgerock.openam.agents.config.conditional.unwanted.http.url.params.regexp[1]=|.*
```

For example, an incoming URL request such as `http://myapp.example.com/customers?country=uk&=bristol&lang=en_GB&area=1343456` that matches a rule such as `myapp.example.com/customers|,`

`coun.*?`, will be cached by the agent as `http://myapp.example.com/customers?=bristol&lang=en_GB`, where both `country` and unnamed parameter are removed and the remaining parameters are sorted alphabetically.

Property: `org.forgerock.openam.agents.config.conditional.unwanted.http.url.params.regexp[n]`

Hot-swap: Yes

Retain Query Parameters (Not yet in the AM console)[4]

Specifies a list of query parameters to be retained for policy decision and caching purposes. The property has the format `[Domain/path] | parameter[,parameter...]` with no spaces between values. Specify values as follows:

- `Domain/path`

  Specifies the incoming request URL. It can take the following values:

  - A domain. For example, `example.com`.

    When you specify a domain, Java agents match both the domain itself and its subdomains. For example, `example.com` matches `mydomain.example.com` and `www.example.com`.

    Domains can also include path information, for example, `example.com/market`, but cannot specify ports.

  - A subdomain. For example, `mydomain.example.com`.

    When you specify a subdomain, Java agents match the domain, the subdomain, and any sub-subdomain. For example, `mydomain.example.com` matches `true.mydomain.example.com`, too.

    Subdomains can include path information, for example, `mydomain.example.com/secure`, but cannot specify ports.

  - A path. For example, `/myapp`.
  - No value, in which case nothing is specified before the | character and the rule applies to every incoming request.

  Note

  It is possible to specify both the Request-Header and Domain/path login formats in an incoming request via the agent properties. If both formats are specified, the Request-Header entries are applied first. If none of the headers in the incoming request match, the Domain/path entries will be applied.

- `parameter[,parameter...]`

  Specifies a comma-separated list of query parameters to retain from the incoming request URL.

  Consider the following constraints when constructing the list of parameters:

- Add a comma (,) character at the beginning or the end of the list to retain all unnamed parameters. For example, `myapp.example.com/customers|,lang` would match both `lang` and any unnamed parameters.
- Add the asterisk (*) character to the list to retain all parameters, including unnamed ones.

The remaining parameters (those that match the list of parameters) are sorted alphabetically.

Examples:

```
org.forgerock.openam.agents.config.conditional.wanted.http.url.params[0]=myapp.example.com/news|area
org.forgerock.openam.agents.config.conditional.wanted.http.url.params[1]=example.com/news|
area,country,location,
```

For example, an incoming URL request such as `http://myapp.example.com/customers?country=uk&=bristol&lang=en_GB&area=1343456` that matches a rule such as `myapp.example.com/customers|,lang`, will be cached by the agent as `http://myapp.example.com/customers?=bristol&lang=en_GB`, where both `lang` and the unnamed parameter are retained and sorted alphabetically.

Property: `org.forgerock.openam.agents.config.conditional.wanted.http.url.params[n]`

Hot-swap: Yes

Regular Expression Retain Query Parameters (Not yet in the AM console)[4]

Specifies a list of regular expressions the agent uses to match query parameters to be retained for policy decision and caching purposes. The property has the format `[Domain/path] | regular_expression[,regular_expression...]` with no spaces between values. Specify values as follows:

- `Domain/path`

  Specifies the incoming request URL. It can take the following values:

  - A domain. For example, `example.com`.

    When you specify a domain, Java agents match both the domain itself and its subdomains. For example, `example.com` matches `mydomain.example.com` and `www.example.com`.

    Domains can also include path information, for example, `example.com/market`, but cannot specify ports.

  - A subdomain. For example, `mydomain.example.com`.

    When you specify a subdomain, Java agents match the domain, the subdomain, and any sub-subdomain. For example, `mydomain.example.com` matches `true.mydomain.example.com`, too.

    Subdomains can include path information, for example, `mydomain.example.com/secure`, but cannot specify ports.

  - A path. For example, `/myapp`.

- No value, in which case nothing is specified before the | character and the rule applies to every incoming request.

> **Note**
>
> It is possible to specify both the Request-Header and Domain/path login formats in an incoming request via the agent properties. If both formats are specified, the Request-Header entries are applied first. If none of the headers in the incoming request match, the Domain/path entries will be applied.

- `regular_expression[,regular_expression...]`

  Specifies a comma-separated list of regular expressions the agent uses to match query parameters to be retained from the incoming request URL.

  Consider the following constraints when constructing your list of regular expressions:

  - Add a comma (,) character at the beginning or the end of the list to retain all unnamed parameters. For example, `myapp.example.com/customers|,lang` would match both `lang` and any unnamed parameters.
  - Consider creating multiple simple regular expressions instead of a single complicated one.

  The remaining parameters (those that match the list of parameters) are sorted alphabetically.

Examples:

```
org.forgerock.openam.agents.config.conditional.wanted.http.url.params.regexp[0]=example.com/market|
regist.*
org.forgerock.openam.agents.config.conditional.wanted.http.url.params.regexp[1]=myapp.example.com/
register|,.*
```

For example, an incoming URL request such as `http://myapp.example.com/customers?country=uk&=bristol&lang=en_GB&area=1343456` that matches a rule such as `myapp.example.com/customers|,coun.*?`, will be cached by the agent as `http://myapp.example.com/customers?=bristol,country=uk`, where both `country` and the unnamed parameter are retained and sorted alphabetically.

Property: `org.forgerock.openam.agents.config.conditional.wanted.http.url.params.regexp[n]`

Hot-swap: Yes

Configuring SSO Properties

This section covers SSO Java agent properties. After creating the agent profile, you access these properties in the AM console under Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > SSO.

This section describes the following property groups:

Cookie Properties

~~Cookie Name~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.iplanet.am.cookie.name`

HTTPOnly Cookie (Not yet in the AM console)[4]

Agents with this property set to `true` mark cookies as HTTPOnly to prevent scripts and third-party programs from accessing the cookies.

Default: `true`

Property: `com.sun.identity.cookie.httponly`

Caching Properties

~~SSO Cache Enable~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console

Property: `com.sun.identity.agents.config.amsso.cache.enable`

Cross-Domain SSO Properties

~~Cross-Domain SSO~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

CDSSO is always enabled.

Property: `com.sun.identity.agents.config.cdsso.enable`

CDSSO Redirect URI

Specifies a URI the Java agent uses to process CDSSO requests.

Default: `/agent_URI/sunwCDSSORedirectURI`

Property: `com.sun.identity.agents.config.cdsso.redirect.uri`

~~CDSSO Servlet URL~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.agents.config.cdsso.cdcservlet.url`

~~CDSSO Clock Skew~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `>com.sun.identity.agents.config.cdsso.clock.skew`

~~CDSSO Trusted ID Provider~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.agents.config.cdsso.trusted.id.provider`

~~CDSSO Secure Enable~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.agents.config.cdsso.secure.enable`

~~CDSSO Domain List~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.agents.config.cdsso.domain[n]`

Cookie Reset Properties

Cookie Reset

When enabled, Java agents reset cookies in the response before redirecting to authentication.

Default: `false`

Property: `com.sun.identity.agents.config.cookie.reset.enable`

Cookie Reset Name List

List of cookies to reset if Cookie Reset is enabled.

Default: not set

Property: `com.sun.identity.agents.config.cookie.reset.name`

Cookie Reset Domain Map

Specifies how names from the Cookie Reset Name List correspond to cookie domain values when the cookie is reset.

Default: not set

Property: `com.sun.identity.agents.config.cookie.reset.domain`

Cookie Reset Path Map

Specifies how names from the Cookie Reset Name List correspond to cookie paths when the cookie is reset.

Default: not set

Property: `com.sun.identity.agents.config.cookie.reset.path`

Configuring AM Services Properties

This section covers AM services' Java agent properties. After creating the agent profile, you access these properties in the AM console under Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > AM Services.

This section describes the following property groups:

- Login URL Properties
- Logout URL Properties
- Authentication Service Properties
- Policy Client Service Properties
- User Data Cache Service Properties
- Session Client Service Properties

Login URL Properties

For more information, see "Redirection and Conditional Redirection".

Allow Custom Login Mode (Not yet in the AM console)[4]

Specifies whether the agent should use the default or the custom login mode when redirecting unauthenticated users.

Before configuring this property, ensure you have read "Redirection and Conditional Redirection".

- `true`. Custom login redirection mode enabled (Non-OIDC compliant login flow). Use with the following properties:
  - OpenAM Login URL (`com.sun.identity.agents.config.login.url`)

- `org.forgerock.openam.agents.config.conditional.custom.login.url`

- `false`. Default login redirection mode enabled (OIDC compliant login flow). Use with the following properties:

  - OpenAM Conditional Login URL (`org.forgerock.openam.agents.config.conditional.login.url`)

Property: `org.forgerock.openam.agents.config.allow.custom.login`

Default: `false`

OpenAM Login URL

When configured, specifies the URL of a custom login page to which the agent redirects incoming users without sufficient credentials so that they can authenticate.

> **Important**
>
> You must add the custom login page to either the not-enforced IP or URI lists.

Before configuring this property, ensure you have read "Redirection and Conditional Redirection".

The login URL has the format *`URL`*`[?realm=`*`realm_name?parameter1=value1`*`&...]`, where:

- `URL` is the custom login page to where the agent redirects the unauthenticated user.
- `[?realm=`*`realm_name&parameter1=value1`*`&...]` specifies optional parameters that the agent will pass to the custom login page, for example, the AM realm where the user should log to.

  You do not need to specify the realm in the login URL if any of the following conditions is true:

  - The custom login page itself sets the `realm` parameter, for example, because it lets the user choose it. In this case, you must ensure the custom login page *always* returns a `realm` parameter to the agent.
  - The realm that the agent is logging the user into has DNS aliases configured in AM.

    AM logs the user into the realm whose DNS alias matches the incoming request URL. For example, an inbound request from the `http://marketplace.example.com` URL logs in the `marketplace` realm if the realm alias is set to `marketplace.example.com`.
  - The users should always log in to the Top Level Realm.

  Even if you decide to specify the realm by default, this parameter can be overwritten by the custom login page if, for example, the user can choose the realm for authentication.

  You can specify as many parameters your custom login pages require.

Example:

```
https://login.example.com/login.jsp?realm=marketplace&param1=value1
```

In some versions of AM you may be able to configure more than one value for this property, but only the first value is honored.

Important

When the agent redirects the user to the custom login page, it appends a `goto` parameter (as configured in the `com.sun.identity.agents.config.redirect.param` property) with the agent's CDSSO endpoint and a `nonce` parameter.

The following is an example of a redirection from the agent to a custom login page:

```
http://login.example.com/login.jsp?realm=marketplace&param1=value1&goto=http
%3A%2F%2Fagent.example.com%3A8020%2Flogin%2Fendpoint%3Fnonce
%3Df2fc384a07b7668e05fc6c26c01edf1bac8a3b55%26realm%3Dmarketplace
```

Note that the `goto` parameter is URL encoded. If the `realm` parameter is configured in the redirection rule, it is also appended to the `goto` parameter.

Once the user has logged in, the custom login page must redirect back to the agent. To avoid redirection loops and login failures, consider the following constraints:

- You must ensure that the custom login page redirects back to the agent using the URL contained in the `goto` parameter, and that the request contains the `nonce` parameter.
- You must set the `realm` parameter in the redirection request to the agent if the users should not log in to AM's Top Level Realm.

  For example, you could use the realm specified in the redirection request from the agent to the custom login pages (if configured in the conditional redirection rule), or the custom login page can let the user chose to which realm authenticate to.

  The following is an example of a redirection from a custom login page to the agent:

  ```
  http://agent.example.com:8020/login/endpoint?
  nonce=f2fc384a07b7668e05fc6c26c01edf1bac8a3b55&realm=marketplace
  ```

  There is one exception; if the realm where the agent should log the user in to has DNS alias configured, AM will log in the user to the realm whose DNS alias matches the incoming request URL. For example, an inbound request from the `http://marketplace.example.com` URL will be logged in to the marketplace realm if the realm alias is set to `marketplace.example.com`, whether there is a `realm` parameter or not.

Default: *AM_URL*/*AM_URI*/UI/Login

Property: `com.sun.identity.agents.config.login.url`

Hot-swap: Yes

OpenAM Conditional Login URL

Conditionally redirect users based on the incoming request header or URL. If the incoming request URL matches a specified request header or domain name, the Java agent redirects the request to an specific URL. That specific URL can be an AM instance, site, or a different website.

Important

When redirecting incoming login requests to a custom login page, you must add it to either the not-enforced IP or URI lists.

Before configuring this property, ensure you have read "Redirection and Conditional Redirection".

If the FQDN Check property (`com.sun.identity.agents.config.fqdn.check.enable`) is enabled, the Java agent iterates through the list of URLs until it finds an appropriate redirect URL that matches the FQDN check values. Otherwise, the Java agent redirects the user to the URL configured in the conditional redirect rules.

Conditional redirects have the format `[Request-Header:value|Domain/path]|[URL][?realm=value&module=value2&service=value3]`, with no spaces between values. Specify values in conditional redirects as follows:

- `Request-Header`

  Specifies the HTTP request header, which can take the following format:

  - A specific case-insensitive HTTP request header, followed by a colon ':', then its value without line breaks. For example, `X-Source:LAN`.

    If the header is defined but with no value, then any value is acceptable. For example, `X-realm1:`.

    The agent processes any rule in alphabetical order starting with header name, then header values in alphabetical order, and then request headers with zero-length values (i.e., no value).

- `Domain/path`

  Specifies the incoming request URL. It can take the following values:

  - A domain. For example, `example.com`.

    When you specify a domain, Java agents match both the domain itself and its subdomains. For example, `example.com` matches `mydomain.example.com` and `www.example.com`.

    Domains can also include path information, for example, `example.com/market`, but cannot specify ports.

  - A subdomain. For example, `mydomain.example.com`.

    When you specify a subdomain, Java agents match the domain, the subdomain, and any sub-subdomain. For example, `mydomain.example.com` matches `true.mydomain.example.com`, too.

    Subdomains can include path information, for example, `mydomain.example.com/secure`, but cannot specify ports.

- A path. For example, `/myapp`.
- No value, in which case nothing is specified before the | character and the rule applies to every incoming request.

> **Note**
>
> It is possible to specify both the Request-Header and Domain/path login formats in an incoming request via the agent properties. If both formats are specified, the Request-Header entries are applied first. If none of the headers in the incoming request match, the Domain/path entries will be applied.

- `URL`

  Specifies the URL to which redirect incoming login requests. The URL may be an AM instance, an AM site, or a website other than AM.

  When redirecting to AM, specify the URL of an AM instance or site in the format *protocol://FQDN[:port]/URI*`/oauth2/authorize`, where the port is optional if it is 80 or 443. For example, `https://openam.example.com/openam/oauth2/authorize`.

  When redirecting to a website other than AM, specify a URL in the format *protocol://FQDN[:port]/URI*, where the port is optional if it is 80 or 443. For example, `https://myweb.example.com/authApp`.

  If the redirection URL is not specified, the Java agent redirects the request to the AM instance or site specified by the following bootstrap properties:

  ```
  com.iplanet.am.server.protocol://com.iplanet.am.server.host:com.iplanet.am.server.port/
  com.iplanet.am.services.deploymentDescriptor
  ```

- `?realm=`*value*

  Specifies the AM realm into which the agent logs users. For example, `?realm=marketplace`.

  You do not need to specify the realm in the login URL if any of the following conditions is true:

  - The custom login page itself sets the `realm` parameter, for example, because it lets the user choose it. In this case, you must ensure the custom login page *always* returns a `realm` parameter to the agent.
  - The realm that the agent is logging the user into has DNS aliases configured in AM.

    AM logs the user into the realm whose DNS alias matches the incoming request URL. For example, an inbound request from the `http://marketplace.example.com` URL logs in the `marketplace` realm if the realm alias is set to `marketplace.example.com`.
  - The users should always log in to the Top Level Realm.

Even if you decide to specify the realm by default, this parameter can be overwritten by the custom login page if, for example, the user can choose the realm for authentication.

- `&module=value2&service=value3&param1=value1...`

  Specifies parameters that can be added to the URL, such as:

  - `module`, which specifies the authentication module the user authenticates against. For example, `?module=myAuthModule`.
  - `service`, which specifies an authentication chain or tree the user authenticates against. For example, `?service=myAuthChain`.
  - Any other parameters your custom login pages require.

  Chain parameters with an **&** character, for example, `realm=value&service=value`.

Important

Java agent requests contain a number of parameters required by AM's `oauth2/authorize` endpoint. You must ensure that the custom login page redirects back to the agent using the URL contained in the `goto` parameter, and that the request contains the following parameters:

- `response_type=id_token`
- `scope=openid`
- `response_mode=form_post`
- `nonce=one_off_code`
- `client_id=agent_profile_name`
- `agent_realm=agent_realm_name`
- `redirect_uri=agent_CDSSO_endpoint`

The following is an example of the call that should reach AM:

```
https://openam.example.com:443/openam/oauth2/authorize
?scope=openid
&response_type=id_token
&agent_realm=%2F
&redirect_uri=http%3A%2F%2Fopenam.example.com%3A9080%2Ffrqa%2FsunwCDSSORedirectURI
&nonce=sf2fc384a07b7668e05fc6c26c01edf1bac8a3b55
&client_id=myJEEAgent
&response_mode=form_post
```

Failure to maintain these parameters when redirecting to AM may cause unexpected problems, such as redirect loops.

You must also set the `realm` parameter in the redirection request made to the agent if users should not log in to AM's Top Level Realm.

For example, you could use the realm specified in the redirection request from the agent to the custom login pages (if configured in the conditional redirection rule), or the custom login page can let the user chose to which realm authenticate to, and then pass the `realm` parameter to the redirection to the agent.

There is one exception; if the realm where the agent should log in the user has DNS alias configured, AM will log in the user to the realm whose DNS alias matches the incoming request URL. For example, an inbound request from the `http://marketplace.example.com` URL will be logged in the marketplace realm if the realm alias is set to `marketplace.example.com`.

Examples using the *Request-Header* Format

The following examples use the *request headers* format to set redirection to the conditional login URL. The first example redirects the user only if the HTTP header contains the field `X-Source` with the value `LAN`. The second example redirects the user only if the HTTP header contains the field `X-Source` with the value `extranet`. The third example redirects the user if the HTTP header contains the field `X-Realm1` with any value.

```
org.forgerock.openam.agents.config.conditional.login.url[0]=X-Source:LAN|?realm=red&domain=LAN
org.forgerock.openam.agents.config.conditional.login.url[1]=X-Source:extranet|?realm=blue
org.forgerock.openam.agents.config.conditional.login.url[2]=X-Realm1:|?realm=green
```

Important

Be careful when specifying request headers with no values as it may lead to unexpected results.

Rules are applied in alphabetical order of header name, then sub-sorted by alphabetical order of header value. Zero length value always sorts last. For example, if you specify a redirection with an `X-Source` header with value `A` and another redirection with the header `X-Domain` with no value, any incoming request with the `X-Domain` header triggers before the `X-Source`, due to alphabetical header precedence.

```
org.forgerock.openam.agents.config.conditional.login.url[0]=X-Domain:A|http://
openam.example.com:8080/conditional/login.jsp?product=A
org.forgerock.openam.agents.config.conditional.login.url[1]=X-Domain:|http://
openam.example.com:8020/extended/conditional/login.jsp?product=Undefined
org.forgerock.openam.agents.config.conditional.login.url[1]=X-Source:A|http://
openam.example.com:8020/conditional/login.jsp?source=A
```

Examples using the *Domain/Path* Format

The following examples use the *Domain/Path* format to set redirection to the conditional login URL. The first example redirects the user if the parent domain is `example.com`. The second example redirects the user if the FQDN is `myapp.domain.com`. The third example redirects the user if the domain and path match `sales.example.com/marketplace`. The last example redirects the user for any incoming request.

```
org.forgerock.openam.agents.config.conditional.login.url[0]=example.com|https://openam.example.com/
openam/oauth2/authorize
org.forgerock.openam.agents.config.conditional.login.url[1]=myapp.domain.com|https://
openam2.example.com/openam/oauth2/authorize?realm=sales
org.forgerock.openam.agents.config.conditional.login.url[2]=sales.example.com/marketplace|?
realm=marketplace
org.forgerock.openam.agents.config.conditional.login.url[3]=|https://openam3.example.com/openam/
oauth2/authorize?realm=customers&module=myAuthModule
```

Default: not set

Property: `org.forgerock.openam.agents.config.conditional.login.url[n]=[Request-Header:value|Domain]|[URL][?realm=value&module=value2&service=value3]`

Hot-swap: Yes

Custom Conditional Login URL (Not yet in the AM console)[4]

Conditionally redirect users based on the incoming request header or URL. If the incoming request URL matches a specified request header or domain name, the Java agent redirects the request to an specific URL. That specific URL can be an AM instance, site, or a different website.

> **Important**
>
> When redirecting incoming login requests to a custom login page, you must add it to either the not-enforced IP or URI lists.

Before configuring this property, ensure you have read "Redirection and Conditional Redirection".

Use this property only if the AM Login URL (`com.sun.identity.agents.config.login.url`) is empty.

If the FQDN Check property (`com.sun.identity.agents.config.fqdn.check.enable`) is enabled, the Java agent iterates through the list of URLs until it finds an appropriate redirect URL that matches the FQDN check values. Otherwise, the Java agent redirects the user to the URL configured in the conditional redirect rules.

Conditional redirects have the format `[Request-Header:value|Domain/path]|[URL?realm=value&parameter1=value1...]`, with no spaces between values. Specify values in conditional redirects as follows:

- `Request-Header`

  Specifies the HTTP request header, which can take the following format:

  - A specific case-insensitive HTTP request header, followed by a colon ':', then its value without line breaks. For example, `X-Source:LAN`.

    If the header is defined but with no value, then any value is acceptable. For example, `X-realm1:`.

    The agent processes any rule in alphabetical order starting with header name, then header values in alphabetical order, and then request headers with zero-length values (i.e., no value).

- `Domain/path`

  Specifies the incoming request URL. It can take the following values:

- A domain. For example, `example.com`.

  When you specify a domain, Java agents match both the domain itself and its subdomains. For example, `example.com` matches `mydomain.example.com` and `www.example.com`.

  Domains can also include path information, for example, `example.com/market`, but cannot specify ports.

- A subdomain. For example, `mydomain.example.com`.

  When you specify a subdomain, Java agents match the domain, the subdomain, and any sub-subdomain. For example, `mydomain.example.com` matches `true.mydomain.example.com`, too.

  Subdomains can include path information, for example, `mydomain.example.com/secure`, but cannot specify ports.

- A path. For example, `/myapp`.

- No value, in which case nothing is specified before the | character and the rule applies to every incoming request.

Note

It is possible to specify both the Request-Header and Domain/path login formats in an incoming request via the agent properties. If both formats are specified, the Request-Header entries are applied first. If none of the headers in the incoming request match, the Domain/path entries will be applied.

- `URL`

  Specifies the URL to which redirect incoming login requests. The URL may be an AM instance, an AM site, or a website other than AM.

  Specify a URL in the format *`protocol://FQDN[:port]/URI`*, where the port is optional if it is 80 or 443. For example:

  ```
  https://myweb.example.com/authApp/login.jsp
  https://openam.example.com:8443/openam/XUI/#login/
  https://openam.example.com:8443/openam/customlogin/login.jsp
  ```

  If the redirection URL is not specified, the Java agent redirects the request to the AM instance or site specified by the following bootstrap properties:

  ```
  com.iplanet.am.server.protocol://com.iplanet.am.server.host:com.iplanet.am.server.port/
  com.iplanet.am.services.deploymentDescriptor
  ```

- `?realm=value`

  Specifies the AM realm into which the agent logs the users. For example, `?realm=marketplace`.

When redirecting to AM's XUI, use an ampersand (**&**) character instead of a question mark (**?**) character. For example, `https://openam.example.com:8443/openam/XUI/#login/&realm=marketplace`

You do not need to specify the realm in the login URL if any of the following conditions is true:

- The custom login page itself sets the `realm` parameter, for example, because it lets the user choose it. In this case, you must ensure the custom login page *always* returns a `realm` parameter to the agent.
- The realm that the agent is logging the user into has DNS aliases configured in AM.

  AM logs the user into the realm whose DNS alias matches the incoming request URL. For example, an inbound request from the `http://marketplace.example.com` URL logs in the `marketplace` realm if the realm alias is set to `marketplace.example.com`.
- The users should always log in to the Top Level Realm.

Even if you decide to specify the realm by default, this parameter can be overwritten by the custom login page if, for example, the user can choose the realm for authentication.

- `&parameter1=value1...`

  Specifies parameters that can be added to the URL. You can add as many parameters as your custom login pages need.

  Chain parameters with an **&** character, for example, `realm=value&parameter1=value1&parameter2=value2`.

Important

When the agent redirects the user to the custom login page, it appends a `goto` parameter (as configured in the `com.sun.identity.agents.config.redirect.param` property) with the agent's CDSSO endpoint and a `nonce` parameter.

The following is an example of a redirection from the agent to a custom login page:

```
http://login.example.com/login.jsp?realm=marketplace&param1=value1&goto=http
%3A%2F%2Fagent.example.com%3A8020%2Flogin%2Fendpoint%3Fnonce
%3Df2fc384a07b7668e05fc6c26c01edf1bac8a3b55%26realm%3Dmarketplace
```

Note that the `goto` parameter is URL encoded. If the `realm` parameter is configured in the redirection rule, it is also appended to the `goto` parameter.

Once the user has logged in, the custom login page must redirect back to the agent. To avoid redirection loops and login failures, consider the following constraints:

- You must ensure that the custom login page redirects back to the agent using the URL contained in the `goto` parameter, and that the request contains the `nonce` parameter.
- You must set the `realm` parameter in the redirection request to the agent if the users should not log in to AM's Top Level Realm.

For example, you could use the realm specified in the redirection request from the agent to the custom login pages (if configured in the conditional redirection rule), or the custom login page can let the user chose to which realm authenticate to.

The following is an example of a redirection from a custom login page to the agent:

```
http://agent.example.com:8020/login/endpoint?
nonce=f2fc384a07b7668e05fc6c26c01edf1bac8a3b55&realm=marketplace
```

There is one exception; if the realm where the agent should log the user in to has DNS alias configured, AM will log in the user to the realm whose DNS alias matches the incoming request URL. For example, an inbound request from the `http://marketplace.example.com` URL will be logged in to the marketplace realm if the realm alias is set to `marketplace.example.com`, whether there is a `realm` parameter or not.

Examples using the *Request-Header* Format

The following examples use the *request headers* format to set redirection to the custom login URL. The first example triggers only if the HTTP header contains the field `X-Domain` with the value `A`. The second example redirects the user if the HTTP header contains the field `X-Domain` with the any value.

```
org.forgerock.openam.agents.config.conditional.custom.login.url[0]=X-Domain:A|http://
openam.example.com:8080/custom/login.jsp?product=A
org.forgerock.openam.agents.config.conditional.custom.login.url[1]=X-Domain:|http://
openam.example.com:8020/extended/custom/login.jsp?product=Undefined
```

Important

Be careful when specifying request headers with no values as it may lead to unexpected results.

Rules are applied in alphabetical order of header name, then sub-sorted by alphabetical order of header value. Zero length value always sorts last. For example, if you specify a redirection with an `X-Source` header with value `A` and another redirection with the header `X-Domain` with no value, any incoming request with the `X-Domain` header triggers before the `X-Source`, due to alphabetical header precedence.

```
org.forgerock.openam.agents.config.conditional.custom.login.url[0]=X-Domain:A|http://
openam.example.com:8080/custom/login.jsp?product=A
org.forgerock.openam.agents.config.conditional.custom.login.url[1]=X-Domain:|http://
openam.example.com:8020/extended/custom/login.jsp?product=Undefined
org.forgerock.openam.agents.config.conditional.custom.login.url[1]=X-Source:A|http://
openam.example.com:8020/custom/login.jsp?source=A
```

Examples using the *Domain/Path* Format

The following examples use the *Domain/Path* format to set redirection to the custom login URL. The first example redirects the user if the parent domain is `example.com`. The second example redirects the user if the FQDN is `myapp.domain.com`. The third example redirects the user if the domain and path match `sales.example.com/marketplace`. The last example redirects the user for any incoming request.

```
org.forgerock.openam.agents.config.conditional.custom.login.url[0]=example.com|https://
openam.example.com/openam/XUI/#login&amp;realm=customers
org.forgerock.openam.agents.config.conditional.custom.login.url[1]=myapp.domain.com|https://
login.example.com/apps/login.jsp?realm=sales
org.forgerock.openam.agents.config.conditional.custom.login.url[2]=sales.example.com/marketplace|?
realm=marketplace
org.forgerock.openam.agents.config.conditional.login.custom.url[3]=|https://login.example.com/apps/
login.jsp?realm=sales&amp;isblue=true&amp;carowner=true
```

Property: `org.forgerock.openam.agents.config.conditional.custom.login.url[n]= [Request-Header:value|Domain/path]|[URL][?realm=value&parameter1=value1]`

Hot-swap: Yes

~~Login URL Prioritized~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.agents.config.login.url.prioritized`

~~Login URL Probe~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.agents.config.login.url.probe.enabled`

~~Login URL Probe Timeout~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.agents.config.login.url.probe.timeout`

Logout URL Properties

~~OpenAM Logout URL~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.agents.config.logout.url`

OpenAM Conditional Logout URL

Conditionally redirect users based on the incoming request header or URL. If the incoming request URL matches a request header or specified domain name, the Java agent redirects the request to an specific URL. That specific URL can be an AM instance, site, or a different website.

If the FQDN Check property (`com.sun.identity.agents.config.fqdn.check.enable`) is enabled, the Java agent iterates through the list of URLs until it finds an appropriate redirect URL that matches the FQDN check values. Otherwise, the Java agent redirects the user to the URL configured in the conditional redirect rules.

Conditional redirects have the format `[Requester-Header:value|Domain/path]|[URL][?realm=value]`, with no spaces between values. Specify values in conditional redirects as follows:

- `Request-Header`

  Specifies the HTTP request header, which can take the following format:

  - A specific case-insensitive HTTP request header, followed by a colon ':', then its value without line breaks. For example, `X-Source:LAN`.

    The agent processes any rule in alphabetical order starting with header name, and then header values in alphabetical order.

- `Domain/path`

  Specifies the incoming request URL. It can take the following values:

  - A domain. For example, `example.com`.

    When you specify a domain, Java agents match both the domain itself and its subdomains. For example, `example.com` matches `mydomain.example.com` and `www.example.com`.

    Domains can also include path information, for example, `example.com/market`, but cannot specify ports.

  - A subdomain. For example, `mydomain.example.com`.

    When you specify a subdomain, Java agents match the domain, the subdomain, and any sub-subdomain. For example, `mydomain.example.com` matches `true.mydomain.example.com`, too.

    Subdomains can include path information, for example, `mydomain.example.com/secure`, but cannot specify ports.

  - A path. For example, `/myapp`.

  - No value, in which case nothing is specified before the | character and the rule applies to every incoming request.

  Note

  It is possible to specify both the Request-Header and Domain/path login formats in an incoming request via the agent properties. If both formats are specified, the Request-Header entries are applied first. If none of the headers in the incoming request match, the Domain/path entries will be applied.

- `URL`

  Specifies the URL to which redirect incoming logout requests. The URL may be an AM instance, an AM site, or a website other than AM.

  When redirecting to AM, specify the URL of an AM instance or site in the format *`protocol://FQDN[:port]/URI`*`/oauth2/authorize`, where the port is optional if it is 80 or 443. For example, `https://openam.example.com/openam/oauth2/authorize`.

When redirecting to a website other than AM, specify a URL in the format *protocol://FQDN[:port]/URI*, where the port is optional if it is 80 or 443. For example, `https://myweb.example.com/authApp`.

If the redirection URL is not specified, the Java agent redirects the request to the AM instance or site specified by the following bootstrap properties:

```
com.iplanet.am.server.protocol://com.iplanet.am.server.host:com.iplanet.am.server.port/
com.iplanet.am.services.deploymentDescriptor
```

- `?realm=value`

  Specifies the realm the user should log out from. For example, `realm=marketplace`.

  The realm can also be specified in the URL. For example, if the user should log out from the `/customers` realm, construct the URL as `https://openam.example.com/openam/oauth2/realms/root/realms/customers/authorize` and do not add the `realm` parameter.

Examples using the *Request-Header* Format

The following examples use the *request headers* format to set redirection to the conditional logout URL. The first example redirects the user only if the HTTP header contains the field `X-Source` with the value `LAN`. The second example redirects the user only if the HTTP header contains the field `X-Source` with the value `extranet`.

```
com.sun.identity.agents.config.logout.uri[0]= X-source:LAN|/examples/lan_logout.html
com.sun.identity.agents.config.logout.uri[1]= X-source:extranet|/banking/extranet_logout.html
```

Examples using the *Domain/Path* Format

The following examples use the *Domain/Path* format to set redirection to the conditional logout URL. The first example redirects the user if the parent domain is `example.com`. The second example redirects the user if the domain and path match `sales.example.com/marketplace`. The last example redirects the user for any incoming request.

```
org.forgerock.openam.agents.config.conditional.logout.url[0]=example.com|https://openam.example.com/
openam/oauth2/authorize
org.forgerock.openam.agents.config.conditional.logout.url[2]=sales.example.com/marketplace|?
realm=marketplace
org.forgerock.openam.agents.config.conditional.logout.url[3]=|https://openam3.example.com/openam/
oauth2/authorize?realm=customers
```

Property: `org.forgerock.openam.agents.config.conditional.logout.url[n]= [Request-Header:value|Domain/path]|[URL][?realm=value]`

Hot-swap: Yes

~~Logout URL Prioritized~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.agents.config.logout.url.prioritized`

~~Logout URL Probe~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.agents.config.logout.url.probe.enabled`

~~Logout URL Probe Timeout~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.agents.config.logout.url.probe.timeout`

Authentication Service Properties

OpenAM Authentication Service Protocol

Specifies the protocol used by the AM authentication service.

Note that this is a bootstrap property. If you need to change it manually, configure it in the `OpenSSOAgentBootstrap.properties` file.

Default: *`AM_PROTOCOL`*

Property: `com.iplanet.am.server.protocol`

Hot-swap: no

OpenAM Authentication Service Host Name

Specifies the AM authentication service host name.

Note that this is a bootstrap property. If you need to change it manually, configure it in the `OpenSSOAgentBootstrap.properties` file.

Default: *`AM_FQDN`*

Property: `com.iplanet.am.server.host`

Hot-swap: no

OpenAM Authentication Service Port

Specifies the AM authentication service port number.

Note that this is a bootstrap property. If you need to change it manually, configure it in the `OpenSSOAgentBootstrap.properties` file.

Default: *`AM_PORT`*

Property: `com.iplanet.am.server.port`

Hot-swap: no

Policy Client Service Properties

Realm

Realm where AM starts policy evaluation for this Java agent.

Edit this property when AM should start policy evaluation in a realm other than the Top Level Realm, `/`, when handling policy decision requests from this Java agent.

This property is recognized by AM, not the agent.

Default: `/` (top-level realm)

Property: `org.forgerock.openam.agents.config.policy.evaluation.realm`

Hot-swap: yes

Application

The name of the policy set where AM looks for policies to evaluate for this Java agent.

Edit this property when AM should look for policies that belong to a policy set other than `iPlanetAMWebAgentService` when handling policy decision requests from this Java agent.

This property is recognized by AM, not the agent.

Default: `iPlanetAMWebAgentService`

Property: `org.forgerock.openam.agents.config.policy.evaluation.application`

Hot-swap: yes

Enable Policy Notifications

Specifies whether AM notifies the Java agent when the administrator changes a policy.

Default: `true`

Property: `com.sun.identity.agents.notification.enabled`

Hot-swap: no

Policy Client Polling Interval

Specifies the time in minutes after which policy cache entries expire.

Default: `3`

Property: `com.sun.identity.agents.polling.interval`

Hot-swap: no

Active Session Cache Timeout (Not yet in the AM console)[4]

Specifies the time interval in minutes after which an active session expires in the Java agent's cache. The session cache holds information about logged in users, such as session properties and profile attributes.

If the value is not set, the Java agent sets the property to five times the value of the `com.sun.identity.agents.polling.interval` property.

Default: `3`

Property: `org.forgerock.openam.agents.config.active.session.cache.ttl.minutes`

Hot-Swap: yes

~~**Policy Client Cache Mode**~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.policy.client.cacheMode`

~~**Policy Client Boolean Action Values**~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.policy.client.booleanActionValues`

~~**Policy Client Resource Comparators**~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.policy.client.resourceComparators`

~~**Policy Client Clock Skew**~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.policy.client.clockSkew`

URL Policy Env GET Parameters

Specifies the list of HTTP GET request parameters whose names and values the Java agent sets in the environment map for URL policy evaluation by the AM server.

Default: not set

Property: `com.sun.identity.agents.config.policy.env.get.param`

URL Policy Env POST Parameters

Specifies the list of HTTP POST request parameters whose names and values the Java agent sets in the environment map for URL policy evaluation by the AM server.

Default: not set

Property: `com.sun.identity.agents.config.policy.env.post.param`

URL Policy Env jsession Parameters

Specifies the list of HTTP session attributes whose names and values the Java agent sets in the environment map for URL policy evaluation by the AM server.

Default: not set

Property: `com.sun.identity.agents.config.policy.env.jsession.param`

~~**Use HTTP-Redirect for composite advice**~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.agents.config.policy.advice.use.redirect`

User Data Cache Service Properties

~~**Enable Notification of User Data Caches**~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.idm.remote.notification.enabled`

~~**User Data Cache Polling Time**~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.iplanet.am.sdk.remote.pollingTime`

~~**Enable Notification of Service Data Caches**~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.sm.notification.enabled`

~~**Service Data Cache Time**~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.sm.cacheTime`

Session Client Service Properties

Enable Client Polling

Specifies whether the Java agent must poll AM to retrieve information about events such as logouts.

If set to `FALSE`, the Java Agent instead subscribes to receive notifications by using websockets.

Default: `FALSE`

Property: `com.iplanet.am.session.client.polling.enable`

Hot-swap: yes

~~**Client Polling Period**~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.iplanet.am.session.client.polling.period`

Configuring Miscellaneous Properties

This section covers miscellaneous Java agent properties. After creating the agent profile, you access these properties in the AM console under Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Miscellaneous.

This section describes the following property groups:

- Locale Properties
- Port Check Processing Properties
- Bypass Principal List Properties
- Agent Password Encryptor Properties
- Ignore Path Info Properties
- Deprecated Agent Properties

Locale Properties

Locale Language

The default language for the agent.

Default: `en`

Property: `com.sun.identity.agents.config.locale.language`

Hot-swap: no

Locale Country

The default country for the agent.

Default: `US`

Property: `com.sun.identity.agents.config.locale.country`

Hot-swap: no

Port Check Processing Properties

Port Check Enable

When enabled, activate port checking, correcting requests on the wrong port.

Default: `false`

Property: `com.sun.identity.agents.config.port.check.enable`

Port Check File

Specifies the name of the file containing the content to handle requests on the wrong port when port checking is enabled.

Default: `PortCheckContent.txt`

Property: `com.sun.identity.agents.config.port.check.file`

Port Check Setting

Specifies which ports correspond to which protocols. The Java agent uses the map when handling requests with invalid port numbers during port checking.

Default: *8080 http*

Property: `com.sun.identity.agents.config.port.check.setting`

Bypass Principal List Properties

~~Bypass Principal List~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.agents.config.bypass.principal`

Agent Password Encryptor Properties

~~Encryption Provider~~

Specifies the class the agent uses for encryption and decryption purposes, for example, `org.forgerock.openam.shared.security.crypto.AESWrapEncryption`.

Default: `com.iplanet.services.util.JCEEncryption`

Property: `com.iplanet.security.encryptor`

Ignore Path Info Properties

Ignore Path Info in Request URL

When enabled, strip the path information from the request URL while doing not-enforced list checks and URL policy evaluation. This property is designed to prevent a user from accessing a URI by appending the matching pattern in the policy or not-enforced list.

For example, if the not-enforced list includes `/*.gif`, then stripping path info from the request URL prevents access to `http://host/index.html` by using `http://host/index.html?hack.gif`.

Default: `false`

Property: `com.sun.identity.agents.config.ignore.path.info`

Deprecated Agent Properties

Goto Parameter Name

Allows you to rename the `goto` parameter. The Java agent appends the requested URL to the renamed parameter during redirection. Rename the parameter when your application requires a parameter other than `goto`.

Consider the following example:

```
com.sun.identity.agents.config.redirect.param=goto2
```

A valid redirection URL using the `goto2` parameter may look similar to the following:

```
https://www.example.com:8443/accessDenied.html?goto2=http%3A%2F%www.example.com%3A8020%managers
%2Findex.jsp
```

In this example, the URL appended to the `goto2` parameter is the URL that the user tried to access when the Java agent redirected the request to the `accessDenied.html` page. Note that you configure the access denied page using the Access Denied URI Processing (`com.sun.identity.agents.config.access.denied.uri`) property.

Default: `goto`

Property: `com.sun.identity.agents.config.redirect.param`

Legacy User Agent Support Enable

When enabled, provide support for legacy browsers.

Default: `false`

Property: `com.sun.identity.agents.config.legacy.support.enable`

Legacy User Agent List

List of header values that identify legacy browsers. Entries can use the wildcard character, `*`.

Default: `Mozilla/4.7*`

Property: `com.sun.identity.agents.config.legacy.user.agent`

Legacy User Agent Redirect URI

Specifies a URI the Java agent uses to redirect legacy user agent requests.

Default: *`agent_URI`*`/sunwLegacySupportURI`

Property: `com.sun.identity.agents.config.legacy.redirect.uri`

Configuring Advanced Properties

This section covers advanced Java agent properties. After creating the agent profile, you access these properties in the AM console under Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced.

This section describes the following property groups:

- Client Identification Properties
- Web Service Processing Properties
- Alternate Agent URL Properties
- JBoss Application Server Properties
- Cross-Site Scripting Detection Properties
- POST Data Preservation Properties
- Custom Properties

Client Identification Properties

If the Java agent is behind a proxy or load balancer, then the Java agent can get client IP and host name values from the proxy or load balancer. For proxies and load balancers that support providing the client IP and host name in HTTP headers, you can use the following properties.

When multiple proxies or load balancers sit in the request path, the header values can include a comma-separated list of values with the first value representing the client, as in `client,next-proxy, first-proxy`.

Client IP Address Header

HTTP header name that holds the IP address of the client.

If the Java agent is behind a proxy or load balancer, then the Java agent can get client IP address values from the proxy or load balancer. Use this property if the proxy or load balancer supports providing the IP address in an HTTP header.

Default: not set

Property: `com.sun.identity.agents.config.client.ip.header`

Client Hostname Header

HTTP header name that holds the hostname of the client.

If the Java agent is behind a proxy or load balancer, then the Java agent can get client host name values from the proxy or load balancer. Use this property if the proxy or load balancer supports providing the host name in an HTTP header.

When multiple proxies or load balancers sit in the request path, the header values can include a comma-separated list of values with the first value representing the client, as in `client,next-proxy, first-proxy`.

Default: not set

Property: `com.sun.identity.agents.config.client.hostname.header`

Web Service Processing Properties

The following properties do not apply to Java Agents 5.6, although they may appear in the AM console:

~~**Web Service Enable**~~

Property: `com.sun.identity.agents.config.webservice.enable`

~~Web Service End Points~~

Property: `com.sun.identity.agents.config.webservice.endpoint`

~~Web Service Process GET Enable~~

Property: `com.sun.identity.agents.config.webservice.process.get.enable`

~~Web Service Authenticator~~

Property: `com.sun.identity.agents.config.webservice.authenticator`

~~Web Service Response Processor~~

Property: `com.sun.identity.agents.config.webservice.responseprocessor`

~~Web Service Internal Error Content File~~

Property: `com.sun.identity.agents.config.webservice.internalerror.content`

~~Web Service Authorization Error Content File~~

Property: `com.sun.identity.agents.config.webservice.autherror.content`

Alternate Agent URL Properties

Alternative Agent Host Name

Specifies the host name of the Java agent protected server to show to client browsers, rather than the actual host name.

After configuring this property, you must add an entry to the Agent Root URL for CDSSO property with the new hostname. Failure to do so may result in redirection loops or `redirect_uri_mismatch` errors.

Default: not set

Property: `com.sun.identity.agents.config.agent.host`

Alternative Agent Port Name

Specifies the port number of the Java agent protected server to show to client browsers, rather than the actual port number.

After configuring this property, you must add an entry to the Agent Root URL for CDSSO property with the new port. Failure to do so may result in redirection loops or `redirect_uri_mismatch` errors.

Default: not set

Property: `com.sun.identity.agents.config.agent.port`

Alternative Agent Protocol

Specifies the protocol used to contact the Java agent from the browser client browsers, rather than the actual protocol used by the server. Either `http` or `https`.

After configuring this property, you must add an entry to the Agent Root URL for CDSSO property with the new protocol. Failure to do so may result in redirection loops or `redirect_uri_mismatch` errors.

Default: not set

Property: `com.sun.identity.agents.config.agent.protocol`

JBoss Application Server Properties

~~WebAuthentication Available~~

This property does not apply to Java Agents 5.6, although it may appear in the AM console.

Property: `com.sun.identity.agents.config.jboss.webauth.available`

Cross-Site Scripting Detection Properties

Possible XSS code elements

Specifies strings that, when found in the request, cause the agent to redirect the client to an error page.

Default: not set

Property: `com.sun.identity.agents.config.xss.code.elements`

XSS detection redirect URI

Maps applications to URIs of customized pages to which to redirect clients upon detection of XSS code elements.

For example, to redirect clients of MyApp to `/myapp/error.html`, enter MyApp as the Map Key and `/myapp/error.html` as the Corresponding Map Value.

Default: not set

Property: `com.sun.identity.agents.config.xss.redirect.uri`

POST Data Preservation Properties

For more information about POST data Preservation, see "POST Data Preservation".

POST Data Preservation enabled

Enables HTTP POST data preservation, storing POST data before redirecting the browser to the login screen, and then auto-submitting the same POST after successful authentication to the original URL.

For more information, see "POST Data Preservation".

Default: `false`

Property: `com.sun.identity.agents.config.postdata.preserve.enable`

Missing PDP entry URI

Specifies a list of application-specific URIs if the referenced Post Data Preservation entry cannot be found in the local cache because it has exceeded its POST entry TTL. Either the Java agent redirects to a URI in this list, or it shows an HTTP 403 Forbidden error.

Default: not set

Property: `com.sun.identity.agents.config.postdata.preserve.cache.noentry.url`

PDP entry TTL

POST data storage lifetime in the PDP cache, in milliseconds.

Default: `300000`

Property: `com.sun.identity.agents.config.postdata.preserve.cache.entry.ttl`

PDP Stickysession mode

Specifies whether to create a cookie, or to append a query string to the URL to assist with sticky load balancing.

Default: `URL`

Property: `com.sun.identity.agents.config.postdata.preserve.stickysession.mode`

PDP Stickysession key-value

Specifies the key-value pair for stickysession mode. For example, a setting of `lb=myserver` either sets an `lb` cookie with `myserver` value, or adds `lb=myserver` to the URL query string.

Default: not set

Property: `com.sun.identity.agents.config.postdata.preserve.stickysession.value`

PDP Maximum Number of Cache Entries

Specifies the maximum number of entries to hold in the PDP cache. Old entries in the cache are discarded if the cache reaches the maximum number of entries.

Limiting the amount of entries in the PDP cache mitigates against DoS attacks. If a malicious user posts a large amount of information to the cache, the cache will grow to the maximum number of entries rather than consume all available memory.

Default: `1000`

Property: `com.sun.identity.agents.config.postdata.preserve.cache.entry.max.entries`

PDP Maximum Cache Size

Specifies the maximum size of the PDP cache in megabytes. Old entries in the cache are discarded if the cache reaches the maximum size.

Limiting the size of the PDP cache mitigates against DoS attacks. If a malicious user posts a large amount of information to the cache, the cache will grow to the maximum size rather than consume all available memory.

> **Note**
>
> If both PDP Cache Maximum Size and PDP Cache Maximum Number of Entries properties are set, the PDP Cache Maximum Number of Entries property is ignored.

Default: `-1` (cache in MBs is not created)

Property: `org.forgerock.openam.agents.config.postdata.preserve.cache.entry.max.total.size.mb`

Custom Properties

Custom Properties

Additional properties to augment the set of properties supported by the Java agent. Custom properties can be specified as follows:

- `customproperty=custom-value1`
- `customlist[0]=customlist-value-0`
- `customlist[1]=customlist-value-1`

- `custommap[key1]=custommap-value-1`
- `custommap[key2]=custommap-value-2`

Tip

Add any property that is not yet in the AM console as a custom property.

Property: `com.sun.identity.agents.config.freeformproperties`

Configuring Agent Authenticators

An *agent authenticator* has read-only access to multiple agent profiles defined in the same realm, typically allowing an agent to read web service agent profiles.

After creating the agent profile, access agent properties in the AM console navigatin to Realms > *Realm Name* > Applications > Agents > Agent Authenticator > *Agent Name*.

Password

Specifies the password the agent uses to connect to AM.

Status

Specifies whether the agent profile is active, and so can be used.

Agent Profiles allowed to Read

Specifies which agent profiles the agent authenticator can read in the realm.

Agent Root URL for CDSSO

Specifies the list of agent root URLs for CDSSO. The valid value is in the format `protocol://hostname:port/` where *protocol* represents the protocol used, such as `http` or `https`, *hostname* represents the host name of the system where the agent resides, and *port* represents the port number on which the agent is installed. The slash following the port number is required.

If your agent system also has virtual host names, add URLs with the virtual host names to this list as well. AM checks that `goto` URLs match one of the agent root URLs for CDSSO.

Property: `sunIdentityServerDeviceKeyValue[n]=protocol://hostname:port/`

Monitoring Reference

This section contains reference information for the metric types, and monitoring metrics.

Monitoring Metric Types

This section describes the monitoring metric types that are available in Java Agents. The available types are:

- Timer
- Gauge
- DistinctCounter

Timer

Metric that combines both rate and duration information.

- Fields

  When using the Common REST interface, the `Timer` metric type has the following fields:

Field	Description
`_id`	The metric ID.
`_type`	The metric type.
`count`	The number of events recorded for this metric.
`total`	The sum of the durations recorded for this metric.
`min`	The minimum duration recorded for this metric.
`max`	The maximum duration recorded for this metric.
`mean`	The mean average duration recorded for this metric.
`stddev`	The standard deviation of durations recorded for this metric.
`duration_units`	The units used for measuring the durations in the metric.
`p50`	50% of the durations recorded are at or below this value.
`p75`	75% of the durations recorded are at or below this value.
`p95`	95% of the durations recorded are at or below this value.
`p98`	98% of the durations recorded are at or below this value.
`p99`	99% of the durations recorded are at or below this value.
`p999`	99.9% of the durations recorded are at or below this value.
`m1_rate`	The one-minute average rate.
`m5_rate`	The five-minute average rate.
`m15_rate`	The fifteen-minute average rate.
`mean_rate`	The average rate.
`rate_units`	The units used for measuring the rate of the metric.

Note

Duration-based values, such as `min`, `max`, and `p50`, are weighted towards newer data. By representing approximately the last five minutes of data, the timers make it easier to see recent changes in behavior, rather than a uniform average of recordings since the server was started.

The following is an example of the `requests.granted.not-enforced` metric from the Common REST endpoint:

```
{
  "_id" : "requests.granted.not-enforced",
  "_type" : "timer",
  "count" : 486,
  "total" : 80.0,
  "min" : 0.0,
  "max" : 1.0,
  "mean" : 0.1905615495053855,
  "stddev" : 0.39274399467782056,
  "duration_units" : "milliseconds",
  "p50" : 0.0,
  "p75" : 0.0,
  "p95" : 1.0,
  "p98" : 1.0,
  "p99" : 1.0,
  "p999" : 1.0,
  "m1_rate" : 0.1819109974890356,
  "m5_rate" : 0.05433445522996721,
  "m15_rate" : 0.03155662103953588,
  "mean_rate" : 0.020858521722211427,
  "rate_units" : "calls/second"
}
```

- Prometheus Fields

  The Prometheus endpoint does not provide rate-based statistics, as rates can be calculated from the time-series data.

  When using the Prometheus interface, the `Timer` metric type has the following fields:

Field	Description
`# TYPE`	The metric ID, and type. Note that the `Timer` metric type is reported as a `Summary` type. Formatted as a comment.
`_count`	The number of events recorded.
`_total`	The sum of the durations recorded.
`{quantile="0.5"}`	50% of the durations are at or below this value.
`{quantile="0.75"}`	75% of the durations are at or below this value.
`{quantile="0.95"}`	95% of the durations are at or below this value.
`{quantile="0.98"}`	98% of the durations are at or below this value.

Field	Description
`{quantile="0.99"}`	99% of the durations are at or below this value.
`{quantile="0.999"}`	99.9% of the durations are at or below this value.

Note

Duration-based quantile values are weighted towards newer data. By representing approximately the last five minutes of data, the timers make it easier to see recent changes in behavior, rather than a uniform average of recordings since the server was started.

The following is an example of the `ja_requests{access=granted,decision=allowed-by-policy}` metric from the Prometheus endpoint:

```
# TYPE ja_requests_seconds summary
ja_requests_seconds{access="granted",decision="allowed-by-policy",quantile="0.5",}
 0.013000000000000001
ja_requests_seconds{access="granted",decision="allowed-by-policy",quantile="0.75",}
 0.022000000000000002
ja_requests_seconds{access="granted",decision="allowed-by-policy",quantile="0.95",}
 0.022000000000000002
ja_requests_seconds{access="granted",decision="allowed-by-policy",quantile="0.98",}
 0.022000000000000002
ja_requests_seconds{access="granted",decision="allowed-by-policy",quantile="0.99",}
 0.022000000000000002
ja_requests_seconds{access="granted",decision="allowed-by-policy",quantile="0.999",}
 1.1380000000000001
ja_requests_count{access="granted",decision="allowed-by-policy",} 7.0
ja_requests_seconds_total{access="granted",decision="allowed-by-policy",} 1.21
```

Gauge

Metric for a numerical value that can increase or decrease. The value for a gauge is calculated when requested, and represents the state of the metric at that specific time.

- Fields

  When using the Common REST interface, the `Timer` metric type has the following fields:

Field	Description
`_id`	The metric ID.
`_type`	The metric type.
`value`	The current value of the metric.

The following is an example of the `jvm.used-memory` metric from the Common REST endpoint:

```
{
  "_id" : "jvm.used-memory",
  "_type" : "gauge",
  "value" : 2.13385216E9
}
```

- Prometheus Fields

  When using the Prometheus interface, the `Timer` metric type has the following fields:

Field	Description
`# TYPE`	The metric ID, and type. Formatted as a comment.
`{Metric ID}`	The current value. Large values may be represented in scientific E-notation.

  The following is an example of the `ja_jvm_used_memory_bytes` metric from the Prometheus endpoint:

```
# TYPE ja_jvm_used_memory_bytes gauge
ja_jvm_used_memory_bytes 1.418723328E9
```

DistinctCounter

Metric providing an estimate of the number of *unique* values recorded.

For example, this could be used to estimate the number of unique users who have authenticated, or unique client IP addresses.

> **Note**
>
> The `DistinctCounter` metric is calculated per instance of AM, and cannot be aggregated across multiple instances to get a site-wide view.

- Fields

  When using the Common REST interface, the `DistinctCounter` metric type has the following fields:

Field	Description
`_id`	The metric ID.
`_type`	The metric type. Note that the `distinctCounter` type is reported as a `gauge` type. The output formats are identical.
`value`	The calculated estimate of the number of unique values recorded in the metric.

  The following is an example of the `authentication.unique-uuid.success` metric from the Common REST endpoint:

```
{
  "_id" : "authentication.unique-uuid.success",
  "_type" : "gauge",
  "value" : 3.0
}
```

- Prometheus Fields

When using the Prometheus interface, the `distinctCounter` metric type has the following fields:

Field	Description
`# TYPE`	The metric ID, and type. Note that the `distinctCounter` type is reported as a `gauge` type. The output formats are identical. Formatted as a comment.
`{Metric ID}`	The calculated estimate of the number of unique values recorded in the metric.

The following is an example of the `ja_notenforced_ip_unmatched_cache_size` metric from the Prometheus endpoint:

```
# TYPE ja_notenforced_ip_unmatched_cache_size gauge
ja_notenforced_ip_unmatched_cache_size 3.0
```

Monitoring Metrics

Java agents expose the monitoring metrics described in this section.

Audit Handler Metrics

Java Agents expose the following audit handler-related monitoring metrics:

`audit.access.generate`

Time taken to generate an audit object. (Timer)

Prometheus name:

```
ja_audit_generate{topic=access}
```

`audit.handler.<handler-type>.default.access.<outcome>`

Time taken to audit outcomes, both locally to the agent and remotely in AM. (Timer)

Prometheus name:

```
ja_audit{handler-type=<handler-type>,name=default,topic=access,outcome=<outcome>}
```

Labels:

`<handler-type>`

`am-delegate`. Remote auditing performed by AM. (Prometheus: `am_delegate`)

`json`. Local audit logging using JSON.

`<outcome>`

`success`

`failure`

Endpoint and REST SDK Metrics

Java Agents expose the following endpoint and REST SDK-related monitoring metrics:

`session-info`

Time taken to retrieve user session information from AM. (Timer)

Prometheus name:

```
ja_session_info
```

`user-profile`

Time taken to retrieve the user profile information from AM. (Timer)

Prometheus name:

```
ja_user_profile
```

`policy-decision`

Time taken to retrieve policy decisions from AM. (Timer)

Prometheus name:

```
ja_policy_decision
```

JSON Web Token (JWT) Metrics

Java Agents expose the following JWT-related monitoring metrics:

`jwt.cache.size`

The size of the JWT cache. (Gauge)

Prometheus name:

```
ja_jwt_cache_size
```

`jwt.cache.eviction`

The eviction count for the JWT cache. (Gauge)

Prometheus name:

```
ja_jwt_cache_eviction
```

`jwt.cache.load-count`

The load count for the JWT cache. (Gauge)

Prometheus name:

```
ja_jwt_cache_load_count
```

`jwt.cache.load-time`

The load time for the JWT cache, in milliseconds. (Gauge)

Prometheus name:

```
ja_jwt_cache_load_time
```

`jwt.cache.hit`

The hit count for the JWT cache. (Gauge)

Prometheus name:

```
ja_jwt_cache{outcome=hit}
```

`jwt.cache.miss`

The miss count for the JWT cache. (Gauge)

Prometheus name:

```
ja_jwt_cache{outcome=miss}
```

JVM Metrics

Java agents expose the JVM-related monitoring metrics covered in this section.

Tip

To get the metric name used by Prometheus, prepend `ja_` to the names below, and replace period (`.`) and hyphen (`-`) characters with underscore (`_`) characters. For example, the `jvm.available-cpus` metric is named `ja_jvm_available_cpus` in Prometheus.

JVM Metrics by Name

Name	Description
`jvm.available-cpus`	Number of processors available to the Java virtual machine. (Gauge)
`jvm.class-loading.loaded`	Number of classes loaded since the Java virtual machine started. (Gauge)
`jvm.class-loading.unloaded`	Number of classes unloaded since the Java virtual machine started. (Gauge)
`jvm.garbage-collector.PS-MarkSweep.count`	Number of collections performed by the "parallel scavenge mark sweep" garbage collection algorithm. (Gauge)
`jvm.garbage-collector.PS-MarkSweep.time`	Approximate accumulated time taken by the "parallel scavenge mark sweep" garbage collection algorithm. (Gauge)

Name	Description
jvm.garbage-collector.PS-Scavenge.count	Number of collections performed by the "parallel scavenge" garbage collection algorithm. (Gauge)
jvm.garbage-collector.PS-Scavenge.time	Approximate accumulated time taken by the "parallel scavenge" garbage collection algorithm. (Gauge)
jvm.memory-usage.heap.init	Amount of heap memory that the Java virtual machine initially requested from the operating system. (Gauge)
jvm.memory-usage.heap.max	Maximum amount of heap memory that the Java virtual machine will attempt to use. (Gauge)
jvm.memory-usage.heap.committed	Amount of heap memory that is committed for the Java virtual machine to use. (Gauge)
jvm.memory-usage.heap.used	Amount of heap memory used by the Java virtual machine. (Gauge)
jvm.memory-usage.total.init	Amount of memory that the Java virtual machine initially requested from the operating system. (Gauge)
jvm.memory-usage.total.max	Maximum amount of memory that the Java virtual machine will attempt to use. (Gauge)
jvm.memory-usage.non-heap.init	Amount of non-heap memory that the Java virtual machine initially requested from the operating system. (Gauge)
jvm.memory-usage.non-heap.max	Maximum amount of non-heap memory that the Java virtual machine will attempt to use. (Gauge)
jvm.memory-usage.non-heap.committed	Amount of non-heap memory that is committed for the Java virtual machine to use. (Gauge)
jvm.memory-usage.non-heap.used	Amount of non-heap memory used by the Java virtual machine. (Gauge)
jvm.memory-usage.pools.Code-Cache.init	Amount of "code cache" memory that the Java virtual machine initially requested from the operating system. (Gauge)
jvm.memory-usage.pools.Code-Cache.max	Maximum amount of "code cache" memory that the Java virtual machine will attempt to use. (Gauge)
jvm.memory-usage.pools.Code-Cache.committed	Amount of "code cache" memory that is committed for the Java virtual machine to use. (Gauge)
jvm.memory-usage.pools.Code-Cache.used	Amount of "code cache" memory used by the Java virtual machine. (Gauge)
jvm.memory-usage.pools.Compressed-Class-Space.init	Amount of "compressed class space" memory that the Java virtual machine initially requested from the operating system. (Gauge)
jvm.memory-usage.pools.Compressed-Class-Space.init	Maximum amount of "compressed class space" memory that the Java virtual machine will attempt to use. (Gauge)

Name	Description
`jvm.memory-usage.pools.Compressed-Class-Space.committed`	Amount of "compressed class space" memory that is committed for the Java virtual machine to use. (Gauge)
`jvm.memory-usage.pools.Compressed-Class-Space.used`	Amount of "compressed class space" memory used by the Java virtual machine. (Gauge)
`jvm.memory-usage.pools.Metaspace.init`	Amount of "metaspace" memory that the Java virtual machine initially requested from the operating system. (Gauge)
`jvm.memory-usage.pools.Metaspace.max`	Maximum amount of "metaspace" memory that the Java virtual machine will attempt to use. (Gauge)
`jvm.memory-usage.pools.Metaspace.committed`	Amount of "metaspace" memory that is committed for the Java virtual machine to use. (Gauge)
`jvm.memory-usage.pools.Metaspace.used`	Amount of "metaspace" memory used by the Java virtual machine. (Gauge)
`jvm.memory-usage.pools.PS-Eden-Space.init`	Amount of "parallel scavenge eden space" memory that the Java virtual machine initially requested from the operating system. (Gauge)
`jvm.memory-usage.pools.PS-Eden-Space.max`	Maximum amount of "parallel scavenge eden space" memory that the Java virtual machine will attempt to use. (Gauge)
`jvm.memory-usage.pools.PS-Eden-Space.committed`	Amount of "parallel scavenge eden space" memory that is committed for the Java virtual machine to use. (Gauge)
`jvm.memory-usage.pools.PS-Eden-Space.used-after-gc`	Amount of "parallel scavenge eden space" memory after the last time garbage collection recycled unused objects in this memory pool. (Gauge)
`jvm.memory-usage.pools.PS-Eden-Space.used`	Amount of "parallel scavenge eden space" memory used by the Java virtual machine. (Gauge)
`jvm.memory-usage.pools.PS-Old-Gen.init`	Amount of "parallel scavenge old generation" memory that the Java virtual machine initially requested from the operating system. (Gauge)
`jvm.memory-usage.pools.PS-Old-Gen.max`	Maximum amount of "parallel scavenge old generation" memory that the Java virtual machine will attempt to use. (Gauge)
`jvm.memory-usage.pools.PS-Old-Gen.committed`	Amount of "parallel scavenge old generation" memory that is committed for the Java virtual machine to use. (Gauge)
`jvm.memory-usage.pools.PS-Old-Gen.used-after-gc`	Amount of "parallel scavenge old generation" memory after the last time garbage collection recycled unused objects in this memory pool. (Gauge)
`jvm.memory-usage.pools.PS-Old-Gen.used`	Amount of "parallel scavenge old generation" memory used by the Java virtual machine. (Gauge)

Name	Description
`jvm.memory-usage.pools.PS-Survivor-Space.init`	Amount of "parallel scavenge survivor space" memory that the Java virtual machine initially requested from the operating system. (Gauge)
`jvm.memory-usage.pools.PS-Survivor-Space.max`	Maximum amount of "parallel scavenge survivor space" memory that the Java virtual machine will attempt to use. (Gauge)
`jvm.memory-usage.pools.PS-Survivor-Space.committed`	Amount of "parallel scavenge survivor space" memory that is committed for the Java virtual machine to use. (Gauge)
`jvm.memory-usage.pools.PS-Survivor-Space.used-after-gc`	Amount of "parallel scavenge survivor space" memory after the last time garbage collection recycled unused objects in this memory pool. (Gauge)
`jvm.memory-usage.pools.PS-Survivor-Space.used`	Amount of "parallel scavenge survivor space" memory used by the Java virtual machine. (Gauge)
`jvm.memory-usage.total.committed`	Amount of memory that is committed for the Java virtual machine to use. (Gauge)
`jvm.memory-usage.total.used`	Amount of memory used by the Java virtual machine. (Gauge)
`jvm.thread-state.blocked.count`	Number of threads in the BLOCKED state. (Gauge)
`jvm.thread-state.count`	Number of live threads including both daemon and non-daemon threads. (Gauge)
`jvm.thread-state.daemon.count`	Number of live daemon threads. (Gauge)
`jvm.thread-state.new.count`	Number of threads in the NEW state. (Gauge)
`jvm.thread-state.runnable.count`	Number of threads in the RUNNABLE state. (Gauge)
`jvm.thread-state.terminated.count`	Number of threads in the TERMINATED state. (Gauge)
`jvm.thread-state.timed_waiting.count`	Number of threads in the TIMED_WAITING state. (Gauge)
`jvm.thread-state.waiting.count`	Number of threads in the WAITING state. (Gauge)

Not Enforced Rule Metrics

Java Agents expose the following not enforced rule-related monitoring metrics:

`notenforced-uri.matched.cache.size`

The size of the not-enforced URI matched cache. (Gauge)

Prometheus name:

```
ja_notenforced_uri_matched_cache_size
```

`notenforced-uri.matched.cache.eviction`

The eviction count for the not-enforced URI matched cache. (Gauge)

Prometheus name:

```
ja_notenforced_uri_matched_cache_eviction
```

`notenforced-uri.matched.cache.load-count`

The load count for the not-enforced URI matched cache. (Gauge)

Prometheus name:

```
ja_notenforced_uri_matched_cache_load_count
```

`notenforced-uri.matched.cache.load-time`

The load time for the not-enforced URI matched cache, in milliseconds. (Gauge)

Prometheus name:

```
ja_notenforced_uri_matched_cache_load_time
```

`notenforced-uri.matched.cache.hit`

The hit count for the not-enforced URI matched cache. (Gauge)

Prometheus name:

```
ja_notenforced_uri_matched_cache{outcome=hit}
```

`notenforced-uri.matched.cache.miss`

The miss count for the not-enforced URI matched cache. (Gauge)

Prometheus name:

```
ja_notenforced_uri_matched_cache{outcome=miss}
```

`notenforced-uri.unmatched.cache.size`

The size of the not-enforced URI unmatched cache. (Gauge)

Prometheus name:

```
ja_notenforced_uri_unmatched_cache_size
```

`notenforced-uri.unmatched.cache.eviction`

The eviction count for the not-enforced URI unmatched cache. (Gauge)

Prometheus name:

```
ja_notenforced_uri_unmatched_cache_eviction
```

notenforced-uri.unmatched.cache.load-count

The load count for the not-enforced URI unmatched cache. (Gauge)

Prometheus name:

```
ja_notenforced_uri_unmatched_cache_load_count
```

notenforced-uri.unmatched.cache.load-time

The load time for the not-enforced URI unmatched cache, in milliseconds. (Gauge)

Prometheus name:

```
ja_notenforced_uri_unmatched_cache_load_time
```

notenforced-uri.unmatched.cache.hit

The hit count for the not-enforced URI unmatched cache. (Gauge)

Prometheus name:

```
ja_notenforced_uri_unmatched_cache{outcome=hit}
```

notenforced-uri.unmatched.cache.miss

The miss count for the not-enforced URI unmatched cache. (Gauge)

Prometheus name:

```
ja_notenforced_uri_unmatched_cache{outcome=miss}
```

notenforced-ip.matched.cache.size

The size of the not-enforced IP matched cache. (Gauge)

Prometheus name:

```
ja_notenforced_ip_matched_cache_size
```

notenforced-ip.matched.cache.eviction

The eviction count for the not-enforced IP matched cache. (Gauge)

Prometheus name:

```
ja_notenforced_ip_matched_cache_eviction
```

notenforced-ip.matched.cache.load-count

The load count for the not-enforced IP matched cache. (Gauge)

Prometheus name:

```
ja_notenforced_ip_matched_cache_load_count
```

`notenforced-ip.matched.cache.load-time`

The load time for the not-enforced IP matched cache, in milliseconds. (Gauge)

Prometheus name:

```
ja_notenforced_ip_matched_cache_load_time
```

`notenforced-ip.matched.cache.hit`

The hit count for the not-enforced IP matched cache. (Gauge)

Prometheus name:

```
ja_notenforced_ip_matched_cache{outcome=hit}
```

`notenforced-ip.matched.cache.miss`

The miss count for the not-enforced IP matched cache. (Gauge)

Prometheus name:

```
ja_notenforced_ip_matched_cache{outcome=miss}
```

`notenforced-ip.unmatched.cache.size`

The size of the not-enforced IP unmatched cache. (Gauge)

Prometheus name:

```
ja_notenforced_ip_unmatched_cache_size
```

`notenforced-ip.unmatched.cache.eviction`

The eviction count for the not-enforced IP unmatched cache. (Gauge)

Prometheus name:

```
ja_notenforced_ip_unmatched_cache_eviction
```

`notenforced-ip.unmatched.cache.load-count`

The load count for the not-enforced IP unmatched cache. (Gauge)

Prometheus name:

```
ja_notenforced_ip_unmatched_cache_load_count
```

`notenforced-ip.unmatched.cache.load-time`

The load time for the not-enforced IP unmatched cache, in milliseconds. (Gauge)

Prometheus name:

```
ja_notenforced_ip_unmatched_cache_load_time
```

`notenforced-ip.unmatched.cache.hit`

The hit count for the not-enforced IP unmatched cache. (Gauge)

Prometheus name:

```
ja_notenforced_ip_unmatched_cache{outcome=hit}
```

`notenforced-ip.unmatched.cache.miss`

The miss count for the not-enforced IP unmatched cache. (Gauge)

Prometheus name:

```
ja_notenforced_ip_unmatched_cache{outcome=miss}
```

`notenforced-compound.matched.cache.size`

The size of the not-enforced compound matched cache. (Gauge)

Prometheus name:

```
ja_notenforced_compound_matched_cache_size
```

`notenforced-compound.matched.cache.eviction`

The eviction count for the not-enforced compound matched cache. (Gauge)

Prometheus name:

```
ja_notenforced_compound_matched_cache_eviction
```

`notenforced-compound.matched.cache.load-count`

The load count for the not-enforced compound matched cache. (Gauge)

Prometheus name:

```
ja_notenforced_compound_matched_cache_load_count
```

`notenforced-compound.matched.cache.load-time`

The load time for the not-enforced compound matched cache, in milliseconds. (Gauge)

Prometheus name:

```
ja_notenforced_compound_matched_cache_load_time
```

`notenforced-compound.matched.cache.hit`

The hit count for the not-enforced compound matched cache. (Gauge)

Prometheus name:

```
ja_notenforced_compound_matched_cache{outcome=hit}
```

`notenforced-compound.matched.cache.miss`

The miss count for the not-enforced compound matched cache. (Gauge)

Prometheus name:

```
ja_notenforced_compound_matched_cache{outcome=miss}
```

`notenforced-compound.unmatched.cache.size`

The size of the not-enforced compound unmatched cache. (Gauge)

Prometheus name:

```
ja_notenforced_compound_unmatched_cache_size
```

`notenforced-compound.unmatched.cache.eviction`

The eviction count for the not-enforced compound unmatched cache. (Gauge)

Prometheus name:

```
ja_notenforced_compound_unmatched_cache_eviction
```

`notenforced-compound.unmatched.cache.load-count`

The load count for the not-enforced compound unmatched cache. (Gauge)

Prometheus name:

```
ja_notenforced_compound_unmatched_cache_load_count
```

`notenforced-compound.unmatched.cache.load-time`

The load time for the not-enforced compound unmatched cache, in milliseconds. (Gauge)

Prometheus name:

```
ja_notenforced_compound_unmatched_cache_load_time
```

`notenforced-compound.unmatched.cache.hit`

The hit count for the not-enforced compound unmatched cache. (Gauge)

Prometheus name:

```
ja_notenforced_compound_unmatched_cache{outcome=hit}
```

`notenforced-compound.unmatched.cache.miss`

The miss count for the not-enforced compound unmatched cache. (Gauge)

Prometheus name:

```
ja_notenforced_compound_unmatched_cache{outcome=miss}
```

Policy Decision Metrics

Java Agents expose the following policy decision-related monitoring metrics:

`policy-decision.cache.size`

The size of the policy decision cache. (Gauge)

Prometheus name:

```
ja_policy_decision_cache_size
```

`policy-decision.cache.eviction`

The eviction count for the policy decision cache. (Gauge)

Prometheus name:

```
ja_policy_decision_cache_eviction
```

`policy-decision.cache.load-count`

The load count for the policy decision cache. (Gauge)

Prometheus name:

```
ja_policy_decision_cache_load_count
```

`policy-decision.cache.load-time`

The load time for the policy decision cache, in milliseconds. (Gauge)

Prometheus name:

```
ja_policy_decision_cache_load_time
```

`policy-decision.cache.hit`

The hit count for the policy decision cache. (Gauge)

Prometheus name:

```
ja_policy_decision_cache{outcome=hit}
```

`policy-decision.cache.miss`

The miss count for the policy decision cache. (Gauge)

Prometheus name:

```
ja_policy_decision_cache{outcome=miss}
```

POST Data Preservation (PDP) Metrics

Java Agents expose the following PDP-related monitoring metrics:

`pdp.cache.size`

The size of the post-data preservation cache. (Gauge)

Prometheus name:

```
ja_pdp_cache_size
```

`pdp.cache.eviction`

The eviction count for the post-data preservation cache. (Gauge)

Prometheus name:

```
ja_pdp_cache_eviction
```

`pdp.cache.load-count`

The load count for the post-data preservation cache. (Gauge)

Prometheus name:

```
ja_pdp_cache_load_count
```

`pdp.cache.load-time`

The load time for the post-data preservation cache, in milliseconds. (Gauge)

Prometheus name:

```
ja_pdp_cache_load_time
```

`pdp.cache.hit`

The hit count for the post-data preservation cache. (Gauge)

Prometheus name:

```
ja_pdp_cache{outcome=hit}
```

`pdp.cache.miss`

The miss count for the post-data preservation cache. (Gauge)

Prometheus name:

```
ja_pdp_cache{outcome=miss}
```

Request Metrics

Java Agents expose the following request-related monitoring metrics:

`requests.<access>.<decision>`

Rate of granted/denied requests and their decision. (Timer)

Prometheus name:

```
ja_requests{access=<access>,decision=<decision>}
```

Labels:

`<access>`

`granted`

`denied`

`<decision>`

`not-enforced`. Request matched a not enforced rule.

`no-valid-token`. Request did not have a valid SSO token or an OpenID Connect JSON Web Token.

`allowed-by-policy`. Request matched a policy, which allowed access.

`denied-by-policy`. Request matched a policy, which denied access.

`am-unavailable`. The AM instance was not reachable.

`agent-exception`. An internal error (exception) occurred within the agent.

Session Information Metrics

Java Agents expose the following session information-related monitoring metrics:

`session-info.cache.size`

The size of the session information cache. (Gauge)

Prometheus name:

```
ja_session_info_cache_size
```

`session-info.cache.eviction`

The eviction count for the session information cache. (Gauge)

Prometheus name:

```
ja_session_info_cache_eviction
```

`session-info.cache.load-count`

The load count for the session information cache. (Gauge)

Prometheus name:

```
ja_session_info_cache_load_count
```

`session-info.cache.load-time`

The load time for the session information cache, in milliseconds. (Gauge)

Prometheus name:

```
ja_session_info_cache_load_time
```

`session-info.cache.hit`

The hit count for the session information cache. (Gauge)

Prometheus name:

```
ja_session_info_cache{outcome=hit}
```

`session-info.cache.miss`

The miss count for the session information cache. (Gauge)

Prometheus name:

```
ja_session_info_cache{outcome=miss}
```

Websocket Metrics

Java Agents expose the following websocket-related monitoring metrics:

`websocket.last-received`

The number of milliseconds since anything was received over the websocket, for example a ping or a notification. (Gauge)

Prometheus name:

```
ja_websocket_last_received
```

`websocket.last-sent`

The number of milliseconds since anything was sent over the websocket. (Gauge)

Prometheus name:

```
ja_websocket_last_sent
```

`websocket.config-change.received`

The number of configuration change notifications received. Note that some may be ignored if the realm or agent name are not applicable. (DistinctCounter)

Prometheus name:

```
ja_websocket_config_change_received
```

`websocket.config-change.processed`

The number of configuration change notifications processed, that were not ignored. (DistinctCounter)

Prometheus name:

```
ja_websocket_config_change_processed
```

`websocket.policy-change.received`

The number of policy change notifications received. Note that some may be ignored if the realm or agent name are not applicable. (DistinctCounter)

Prometheus name:

```
ja_websocket_policy_change_received
```

`websocket.policy-change.processed`

The number of policy change notifications processed, that were not ignored. (DistinctCounter)

Prometheus name:

```
ja_websocket_policy_change_processed
```

`websocket.session-logout.received`

The number of session logout notifications received. Note that some may be ignored if the realm or agent name are not applicable. (DistinctCounter)

Prometheus name:

```
ja_websocket_session_logout_received
```

`websocket.session-logout.processed`

The number of session logout notifications processed, that were not ignored. (DistinctCounter)

Prometheus name:

```
ja_websocket_session_logout_processed
```

websocket.ping-pong

The ping/pong round trip time. (Timer)

Prometheus name:

```
ja_websocket_ping_pong
```

Command-Line Tool Reference

Name

agentadmin — manage the installation of Java agents

Synopsis

`agentadmin` {options}

Description

This command manages Java Agent installations. The **agentadmin** command requires a Java runtime environment.

Options

The following options are supported.

`--install`

Installs a new agent instance.

Usage: **agentadmin --install [--useResponse | --saveResponse *file-name*]**

`--useResponse`

Use this option to install in silent mode by specifying all the responses in the *file-name* file. When this option is used, **agentadmin** runs in non-interactive mode.

`--saveResponse`

Use this option to save all the supplied responses in a response file specified by *file-name*.

`--custom-install`

Installs a new agent instance, specifying additional configuration options such as the key used to encrypt passwords.

Usage: **agentadmin --custom-install [--useResponse | --saveResponse *file-name*]**

`--useResponse`

Use this option to install in silent mode by specifying all the responses in the *file-name* file. When this option is used, **agentadmin** runs in non-interactive mode.

`--saveResponse`

Use this option to save all the supplied responses to the *file-name* file.

`--acceptLicense`

Auto-accepts the software license agreement. If this option is present on the command line with the `--install` or `--custom-install` option, the license agreement prompt is suppressed and the agent installation continues. To view the license agreement, open `<server-root>/legal-notices/license.txt`.

`--uninstall`

Uninstalls an existing agent instance.

Usage: **agentadmin --uninstall [--useResponse | --saveResponse *file-name*]**

`--useResponse`

Use this option to uninstall in silent mode by specifying all the responses in the *file-name* file. When this option is used, **agentadmin** runs in non-interactive mode.

`--saveResponse`

Use this option to save all the supplied responses to the *file-name* file.

`--version`

Displays the version information.

`--uninstallAll`

Uninstalls all the agent instances.

`--listAgents`

Displays details of all the configured agents.

`--agentInfo`

Displays details of the agent corresponding to the specified *agent-id*.

Example: **agentadmin --agentInfo agent_001**

`--encrypt`

Encrypts a given string.

Usage: **agentadmin --encrypt *agent-instance password-file***

agent-instance

Agent instance identifier. The encryption functionality requires the use of agent instance specific encryption key present in its configuration file.

password-file

File containing the password to encrypt.

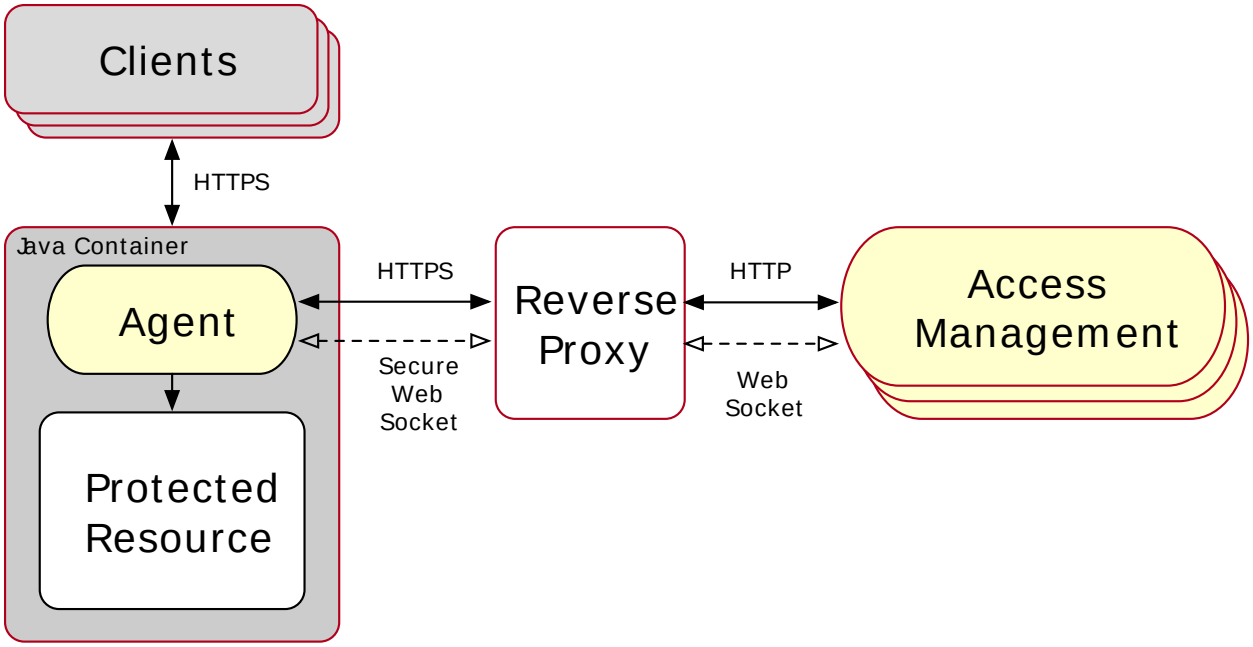
`--getEncryptKey`

Generates an agent encryption key.

Configuring Apache HTTP Server as a Reverse Proxy Example

This section demonstrates a possible configuration of Apache as a reverse proxy between AM and the agent, but you can use any reverse proxy that supports the WebSocket protocol.

Reverse Proxy Configured Between the Agent and AM



Note that the communication protocol changes from HTTPS to HTTP.

To Configure Apache as a Reverse Proxy Example

This procedure demonstrates how to configure Apache as a reverse proxy between an agent and a single AM instance. Refer to the Apache documentation to configure Apache for load balancing and any other requirement for your environment:

1. Locate the `httpd.conf` file in your deployed reverse proxy instance.

2. Add the modules required for a proxy configuration as follows:

```
# Modules required for proxy
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
LoadModule proxy_wstunnel_module modules/mod_proxy_wstunnel.so
```

The `mod_proxy_wstunnel.so` module is required to support the WebSocket protocol used for notification between AM and the agents.

3. Add the proxy configuration inside the `VirtualHost` context. Consider the following directives:

```
<VirtualHost 192.168.1.1>
...
# Proxy Config
RequestHeader set X-Forwarded-Proto "https" ❶
ProxyPass "/openam/notifications" "ws://openam.example.com:8080/openam/notifications"
 Upgrade=websocket ❷
ProxyPass "/openam" "http://openam.example.com:8080/openam" ❸
ProxyPassReverseCookieDomain "openam.internal.example.com" "proxy.example.com" ❹
ProxyPassReverse "/openam" "http://openam.example.com:8080/openam" ❺
...
</VirtualHost>
```

Key:

❶ **RequestHeader**. Set this directive to `https` or `http` depending on the proxy configuration. If the proxy is configured for https, as in the example depicted in the diagram above, set the directive to `https`. Otherwise, set it to `http`.

In a future step you will configure AM to recognize the forwarded header and use it in the `goto` parameter for redirecting back to the agent after authentication.

❷ **ProxyPass**. Set this directive to allow WebSocket traffic between AM and the agent.

If you have HTTPS configured between the proxy and AM, set the directive to use the `wss` protocol instead of `ws`.

❸ **ProxyPass**. Set this directive to allow HTTP traffic between AM and the agent.

❹ **ProxyPassReverseCookieDomain**. Set this directive to rewrite the domain string in `Set-Cookie` headers in the format *internal domain* (AM's domain) *public domain* (proxy's domain).

❺ **ProxyPassReverse**. Set this directive to the same value configured for the `ProxyPass` directive.

For more information about configuring Apache as a reverse proxy, refer to the Apache documentation.

4. Restart the reverse proxy instance.

5. Configure AM to recover the forwarded header you configured in the reverse proxy. Also, review other configurations that may be required in an environment that uses reverse proxies. For more information, see "Regarding Communication Between AM and Agents"

Appendix A. Getting Support

ForgeRock provides support services, professional services, training through ForgeRock University, and partner services to assist you in setting up and maintaining your deployments. For a general overview of these services, see https://www.forgerock.com.

ForgeRock has staff members around the globe who support our international customers and partners. For details on ForgeRock's support offering, including support plans and service level agreements (SLAs), visit https://www.forgerock.com/support.

ForgeRock publishes comprehensive documentation online:

- The ForgeRock Knowledge Base offers a large and increasing number of up-to-date, practical articles that help you deploy and manage ForgeRock software.

  While many articles are visible to community members, ForgeRock customers have access to much more, including advanced information for customers using ForgeRock software in a mission-critical capacity.

- ForgeRock product documentation, such as this document, aims to be technically accurate and complete with respect to the software documented. It is visible to everyone and covers all product features and examples of how to use them.

Glossary

Access control	Control to grant or to deny access to a resource.
Account lockout	The act of making an account temporarily or permanently inactive after successive authentication failures.
Actions	Defined as part of policies, these verbs indicate what authorized identities can do to resources.
Advice	In the context of a policy decision denying access, a hint to the policy enforcement point about remedial action to take that could result in a decision allowing access.
Agent administrator	User having privileges only to read and write agent profile configuration information, typically created to delegate agent profile creation to the user installing a web or Java agent.
Agent authenticator	Entity with read-only access to multiple agent profiles defined in the same realm; allows an agent to read web service profiles.
Application	In general terms, a service exposing protected resources. In the context of AM policies, the application is a template that constrains the policies that govern access to protected resources. An application can have zero or more policies.
Application type	Application types act as templates for creating policy applications. Application types define a preset list of actions and functional logic, such as policy lookup and resource comparator logic.

	Application types also define the internal normalization, indexing logic, and comparator logic for applications.
Attribute-based access control (ABAC)	Access control that is based on attributes of a user, such as how old a user is or whether the user is a paying customer.
Authentication	The act of confirming the identity of a principal.
Authentication chaining	A series of authentication modules configured together which a principal must negotiate as configured in order to authenticate successfully.
Authentication level	Positive integer associated with an authentication module, usually used to require success with more stringent authentication measures when requesting resources requiring special protection.
Authentication module	AM authentication unit that handles one way of obtaining and verifying credentials.
Authorization	The act of determining whether to grant or to deny a principal access to a resource.
Authorization Server	In OAuth 2.0, issues access tokens to the client after authenticating a resource owner and confirming that the owner authorizes the client to access the protected resource. AM can play this role in the OAuth 2.0 authorization framework.
Auto-federation	Arrangement to federate a principal's identity automatically based on a common attribute value shared across the principal's profiles at different providers.
Bulk federation	Batch job permanently federating user profiles between a service provider and an identity provider based on a list of matched user identifiers that exist on both providers.
Circle of trust	Group of providers, including at least one identity provider, who have agreed to trust each other to participate in a SAML v2.0 provider federation.
Client	In OAuth 2.0, requests protected web resources on behalf of the resource owner given the owner's authorization. AM can play this role in the OAuth 2.0 authorization framework.
Client-based OAuth 2.0 tokens	After a successful OAuth 2.0 grant flow, AM returns a token to the client. This differs from CTS-based OAuth 2.0 tokens, where AM returns a *reference* to token to the client.
Client-based sessions	AM sessions for which AM returns session state to the client after each request, and require it to be passed in with the subsequent

	request. For browser-based clients, AM sets a cookie in the browser that contains the session information. For browser-based clients, AM sets a cookie in the browser that contains the session state. When the browser transmits the cookie back to AM, AM decodes the session state from the cookie.
Conditions	Defined as part of policies, these determine the circumstances under which which a policy applies. Environmental conditions reflect circumstances like the client IP address, time of day, how the subject authenticated, or the authentication level achieved. Subject conditions reflect characteristics of the subject like whether the subject authenticated, the identity of the subject, or claims in the subject's JWT.
Configuration datastore	LDAP directory service holding AM configuration data.
Cross-domain single sign-on (CDSSO)	AM capability allowing single sign-on across different DNS domains.
CTS-based OAuth 2.0 tokens	After a successful OAuth 2.0 grant flow, AM returns a *reference* to the token to the client, rather than the token itself. This differs from client-based OAuth 2.0 tokens, where AM returns the entire token to the client.
CTS-based sessions	AM sessions that reside in the Core Token Service's token store. CTS-based sessions might also be cached in memory on one or more AM servers. AM tracks these sessions in order to handle events like logout and timeout, to permit session constraints, and to notify applications involved in SSO when a session ends.
Delegation	Granting users administrative privileges with AM.
Entitlement	Decision that defines which resource names can and cannot be accessed for a given identity in the context of a particular application, which actions are allowed and which are denied, and any related advice and attributes.
Extended metadata	Federation configuration information specific to AM.
Extensible Access Control Markup Language (XACML)	Standard, XML-based access control policy language, including a processing model for making authorization decisions based on policies.
Federation	Standardized means for aggregating identities, sharing authentication and authorization data information between trusted providers, and

	allowing principals to access services across different providers without authenticating repeatedly.
Fedlet	Service provider application capable of participating in a circle of trust and allowing federation without installing all of AM on the service provider side; AM lets you create Java Fedlets.
Hot swappable	Refers to configuration properties for which changes can take effect without restarting the container where AM runs.
Identity	Set of data that uniquely describes a person or a thing such as a device or an application.
Identity federation	Linking of a principal's identity across multiple providers.
Identity provider (IdP)	Entity that produces assertions about a principal (such as how and when a principal authenticated, or that the principal's profile has a specified attribute value).
Identity repository	Data store holding user profiles and group information; different identity repositories can be defined for different realms.
Java agent	Java web application installed in a web container that acts as a policy enforcement point, filtering requests to other applications in the container with policies based on application resource URLs.
Metadata	Federation configuration information for a provider.
Policy	Set of rules that define who is granted access to a protected resource when, how, and under what conditions.
Policy agent	Java, web, or custom agent that intercepts requests for resources, directs principals to AM for authentication, and enforces policy decisions from AM.
Policy Administration Point (PAP)	Entity that manages and stores policy definitions.
Policy Decision Point (PDP)	Entity that evaluates access rights and then issues authorization decisions.
Policy Enforcement Point (PEP)	Entity that intercepts a request for a resource and then enforces policy decisions from a PDP.
Policy Information Point (PIP)	Entity that provides extra information, such as user profile attributes that a PDP needs in order to make a decision.
Principal	Represents an entity that has been authenticated (such as a user, a device, or an application), and thus is distinguished from other entities.

	When a Subject successfully authenticates, AM associates the Subject with the Principal.
Privilege	In the context of delegated administration, a set of administrative tasks that can be performed by specified identities in a given realm.
Provider federation	Agreement among providers to participate in a circle of trust.
Realm	AM unit for organizing configuration and identity information. Realms can be used for example when different parts of an organization have different applications and identity stores, and when different organizations use the same AM deployment. Administrators can delegate realm administration. The administrator assigns administrative privileges to users, allowing them to perform administrative tasks within the realm.
Resource	Something a user can access over the network such as a web page. Defined as part of policies, these can include wildcards in order to match multiple actual resources.
Resource owner	In OAuth 2.0, entity who can authorize access to protected web resources, such as an end user.
Resource server	In OAuth 2.0, server hosting protected web resources, capable of handling access tokens to respond to requests for such resources.
Response attributes	Defined as part of policies, these allow AM to return additional information in the form of "attributes" with the response to a policy decision.
Role based access control (RBAC)	Access control that is based on whether a user has been granted a set of permissions (a role).
Security Assertion Markup Language (SAML)	Standard, XML-based language for exchanging authentication and authorization data between identity providers and service providers.
Service provider (SP)	Entity that consumes assertions about a principal (and provides a service that the principal is trying to access).
Authentication Session	The interval while the user or entity is authenticating to AM.
Session	The interval that starts after the user has authenticated and ends when the user logs out, or when their session is terminated. For browser-based clients, AM manages user sessions across one or more applications by setting a session cookie. See also CTS-based sessions and Client-based sessions.

Session high availability	Capability that lets any AM server in a clustered deployment access shared, persistent information about users' sessions from the CTS token store. The user does not need to log in again unless the entire deployment goes down.
Session token	Unique identifier issued by AM after successful authentication. For a CTS-based sessions, the session token is used to track a principal's session.
Single log out (SLO)	Capability allowing a principal to end a session once, thereby ending her session across multiple applications.
Single sign-on (SSO)	Capability allowing a principal to authenticate once and gain access to multiple applications without authenticating again.
Site	Group of AM servers configured the same way, accessed through a load balancer layer. The load balancer handles failover to provide service-level availability. The load balancer can also be used to protect AM services.
Standard metadata	Standard federation configuration information that you can share with other access management software.
Stateless Service	Stateless services do not store any data locally to the service. When the service requires data to perform any action, it requests it from a data store. For example, a stateless authentication service stores session state for logged-in users in a database. This way, any server in the deployment can recover the session from the database and service requests for any user. All AM services are stateless unless otherwise specified. See also Client-based sessions and CTS-based sessions.
Subject	Entity that requests access to a resource When an identity successfully authenticates, AM associates the identity with the Principal that distinguishes it from other identities. An identity can be associated with multiple principals.
Identity store	Data storage service holding principals' profiles; underlying storage can be an LDAP directory service or a custom `IdRepo` implementation.
Web Agent	Native library installed in a web server that acts as a policy enforcement point with policies based on web page URLs.