



User Guide

/Java Agents 5.7

Latest update: 5.7

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

Copyright © 2011-2020 ForgeRock AS.

Abstract

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



This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License.

To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-nd/3.0/> or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

ForgeRock® and ForgeRock Identity Platform™ are trademarks of ForgeRock Inc. or its subsidiaries in the U.S. and in other countries. Trademarks are the property of their respective owners.

UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

DejaVu Fonts

Bitstream Vera Fonts Copyright

Copyright (c) 2003 by Bitstream, Inc. All Rights Reserved. Bitstream Vera is a trademark of Bitstream, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of the fonts accompanying this license ("Fonts") and associated documentation files (the "Font Software"), to reproduce and distribute the Font Software, including without limitation the rights to use, copy, merge, publish, distribute, and/or sell copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so, subject to the following conditions:

The above copyright and trademark notices and this permission notice shall be included in all copies of one or more of the Font Software typefaces.

The Font Software may be modified, altered, or added to, and in particular the designs of glyphs or characters in the Fonts may be modified and additional glyphs or characters may be added to the Fonts, only if the fonts are renamed to names not containing either the words "Bitstream" or the word "Vera".

This License becomes null and void to the extent applicable to Fonts or Font Software that has been modified and is distributed under the "Bitstream Vera" names.

The Font Software may be sold as part of a larger software package but no copy of one or more of the Font Software typefaces may be sold by itself.

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL BITSTREAM OR THE GNOME FOUNDATION BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Except as contained in this notice, the names of Gnome, the Gnome Foundation, and Bitstream Inc., shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Font Software without prior written authorization from the Gnome Foundation or Bitstream Inc., respectively. For further information, contact: fonts at gnome dot org.

Arev Fonts Copyright

Copyright (c) 2006 by Tavmjong Bah. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of the fonts accompanying this license ("Fonts") and associated documentation files (the "Font Software"), to reproduce and distribute the modifications to the Bitstream Vera Font Software, including without limitation the rights to use, copy, merge, publish, distribute, and/or sell copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so, subject to the following conditions:

The above copyright and trademark notices and this permission notice shall be included in all copies of one or more of the Font Software typefaces.

The Font Software may be modified, altered, or added to, and in particular the designs of glyphs or characters in the Fonts may be modified and additional glyphs or characters may be added to the Fonts, only if the fonts are renamed to names not containing either the words "Tavmjong Bah" or the word "Arev".

This License becomes null and void to the extent applicable to Fonts or Font Software that has been modified and is distributed under the "Tavmjong Bah Arev" names.

The Font Software may be sold as part of a larger software package but no copy of one or more of the Font Software typefaces may be sold by itself.

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL TAVMJONG BAH BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Except as contained in this notice, the name of Tavmjong Bah shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Font Software without prior written authorization from Tavmjong Bah. For further information, contact: tavmjong @ free . fr.

FontAwesome Copyright

Copyright (c) 2017 by Dave Gandy, <https://fontawesome.com/>.

This Font Software is licensed under the SIL Open Font License, Version 1.1. See <https://opensource.org/licenses/OFL-1.1>.

Table of Contents

Preface	iv
1. Introducing Java Agents	1
Java Agent Components	1
Configuration	2
Request Process Flow	4
Java Agent Features	8
2. Preparing for Installation	22
Downloading and Unzipping Java Agents	22
Configuring AM Servers to Communicate With Java Agents	23
Creating Agent Profiles	25
Supporting Load Balancers and Reverse Proxies Between AM and the Agents	27
3. Configuring Environments With Load Balancers and Reverse Proxies	29
Regarding Communication Between AM and Agents	31
Regarding Communication Between Clients and Agents	33
4. Installing Java Agents	40
Installing the Tomcat Java Agent	40
Installing the JBoss Java Agent	47
Installing the Jetty Java Agent	54
Installing the WebLogic Java Agent	61
Installing the WebSphere Java Agent	69
5. Post-Installation Tasks	78
Configuring the Agent Filter	78
Configuring Audit Logging	81
Configuring Performance Monitoring	84
Configuring Java Agents for SSL Communication	86
Supporting Load Balancers and Reverse Proxies Between Clients and Agents	88
6. Upgrading Java Agents	89
7. Removing Java Agents	91
Removing the Tomcat Java Agent	91
Removing the JBoss Java Agent	92
Removing the Jetty Java Agent	94
Removing the WebLogic Java Agent	95
Removing the WebSphere Java Agent	97
8. Troubleshooting	100
9. Reference	103
Configuring Java Agent Properties	103
Configuring Agent Authenticators	207
Monitoring Reference	207
Command-Line Tool Reference	229
Configuring Apache HTTP Server as a Reverse Proxy Example	232
Implementing Custom Task Handlers	234
Glossary	240

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"
- "Configuration"
- "Request Process Flow"
- "Java Agent Features"

Java Agent Components

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

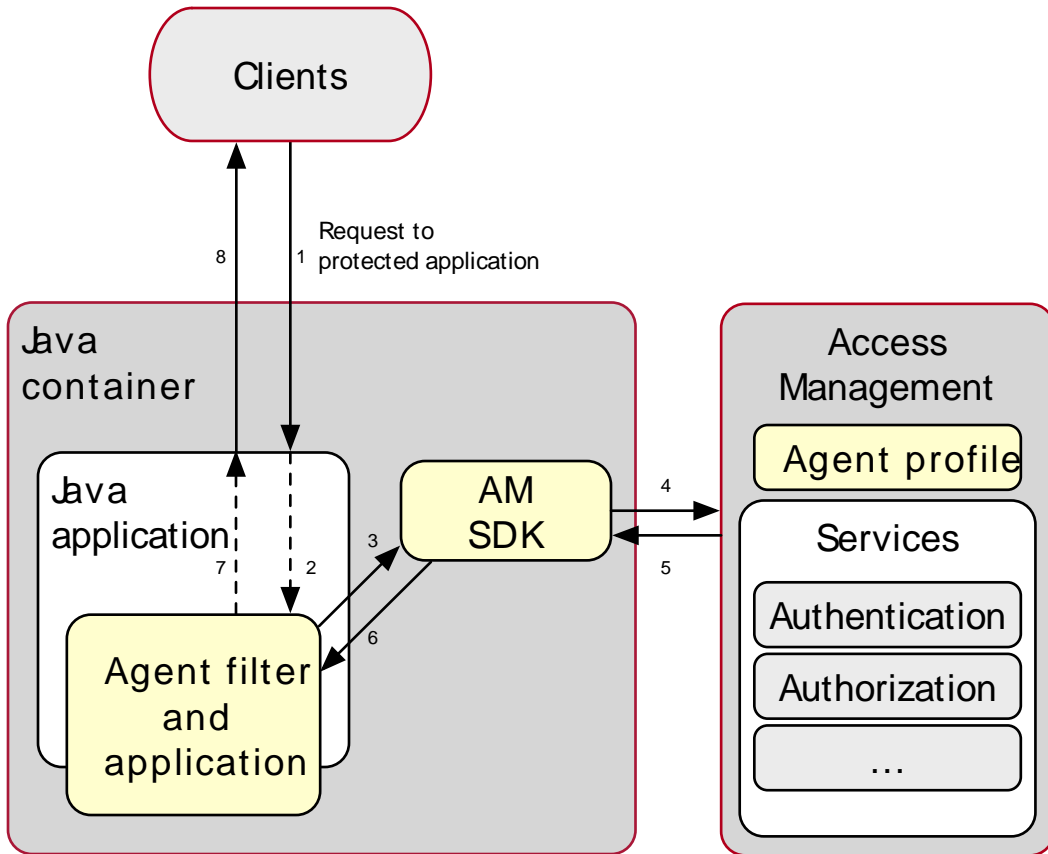
- **Agent Filter.** Intercepts inbound client requests to a resource and processes them based on the filter mode of operation.
- **Agent Application.** Deployed as `agentapp.war`, it is required for authentication and the cross-domain single sign-on (CDSSO) flow.

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

On startup, the Java Agent reads its configuration from the bootstrap properties file `OpenSSOAgentBootstrap.properties`.

The agent finds the location of the bootstrap file from a property added to `JAVA_OPTS`. For example, in Tomcat, it can take the file location from `bin/setenv.sh` as follows:

```
JAVA_OPTS="$JAVA_OPTS -Dopenam.agents.bootstrap.dir=/path/to/java_agents/java_agent/tomcat_agent/config"
```

The Agent has two modes of operation, `LOCAL` and `REMOTE`, which determine the location of the agent configuration. The mode is defined by the property `Configuration Repository`, as follows:

- LOCAL mode, `org.forgerock.agents.config.location=LOCAL`

The Agent reads its configuration from `OpenSSOAgentConfiguration.properties`, found in the same directory as `OpenSSOAgentBootstrap.properties`.

Depending on the settings you add to the configuration file (for example, if `org.forgerock.agents.fallback.mode.enabled` is enabled), the Agent might never need to contact AM.

While the Agent is in LOCAL mode, it ignores any changes made to the Agent profile within AM.

- REMOTE mode, `org.forgerock.agents.config.location=REMOTE`

This is default mode; you don't need to explicitly specify it.

When the first user request is made, the Agent contacts AM to retrieve the Agent configuration. The Agent ignores the configuration in `OpenSSOAgentConfiguration.properties`, and any changes made to it.

The Agent waits until the first user request before it contacts AM, so that AM does not have to be available when the Agent is started. This causes a delay on the first request, but not on subsequent requests.

If AM is unavailable, the request returns an `HTTP 403 Permission denied`. The message written to the debugging logs depends on your logging level.

Bootstrap Properties

The Agent expects bootstrap information on startup in order to contact AM, even if the Agent configuration dictates that the Agent never contacts AM

The following properties are required, and provided by the installer during installation:

- AM Private URL

Used by the Agent for direct communication with AM, for example, to retrieve policy information or user information. The URL is assembled from the following properties:

- AM Protocol (`org.forgerock.agents.am.protocol`)
- AM Host (`org.forgerock.agents.am.hostname`)
- AM Port (`org.forgerock.agents.am.port`)
- AM Path (`org.forgerock.agents.am.path`)
- Java Agent's profile name (`org.forgerock.agents.profile.name`)
- Java Agent's profile realm (`org.forgerock.agents.profile.realm`)
- Java Agent's profile password (`org.forgerock.agents.encrypted.password`)

- Encryption key used for the profile password (`am.encryption.pwd`)

The following property is optional, but must be provided by the user if the AM firewall rules distinguish between public and private URL.

- AM Public URL (`org.forgerock.agents.public.am.url`)

Used by the Agent to redirect the user's browser to public facing URLs for login. If not provided, the AM private URL is used.

Changing configuration Properties

Change the agent configuration in the following ways:

- Changing Agent bootstrap configuration

To change bootstrap properties, manually edit `OpenSSOAgentBootstrap.properties`, and then restart the container running the Java Agent.

- Changing Agent configuration in LOCAL mode

The `OpenSSOAgentConfiguration.properties` is used only in LOCAL mode to set Agent configuration properties. To update properties in LOCAL mode, manually edit the configuration file, and set a value for Configuration Reload Interval.

The interval defines the number of seconds after which the Java Agent reads the local property file, and reloads it if has changed since it was last read.

- Changing Agent configuration in REMOTE mode

In the default REMOTE mode, the Agent is notified by the WebSocket mechanism when its configuration is changed in AM. The Agent then re-reads its configuration from AM within a few seconds.

To change the Agent configuration in the AM console, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name*.

Request Process Flow

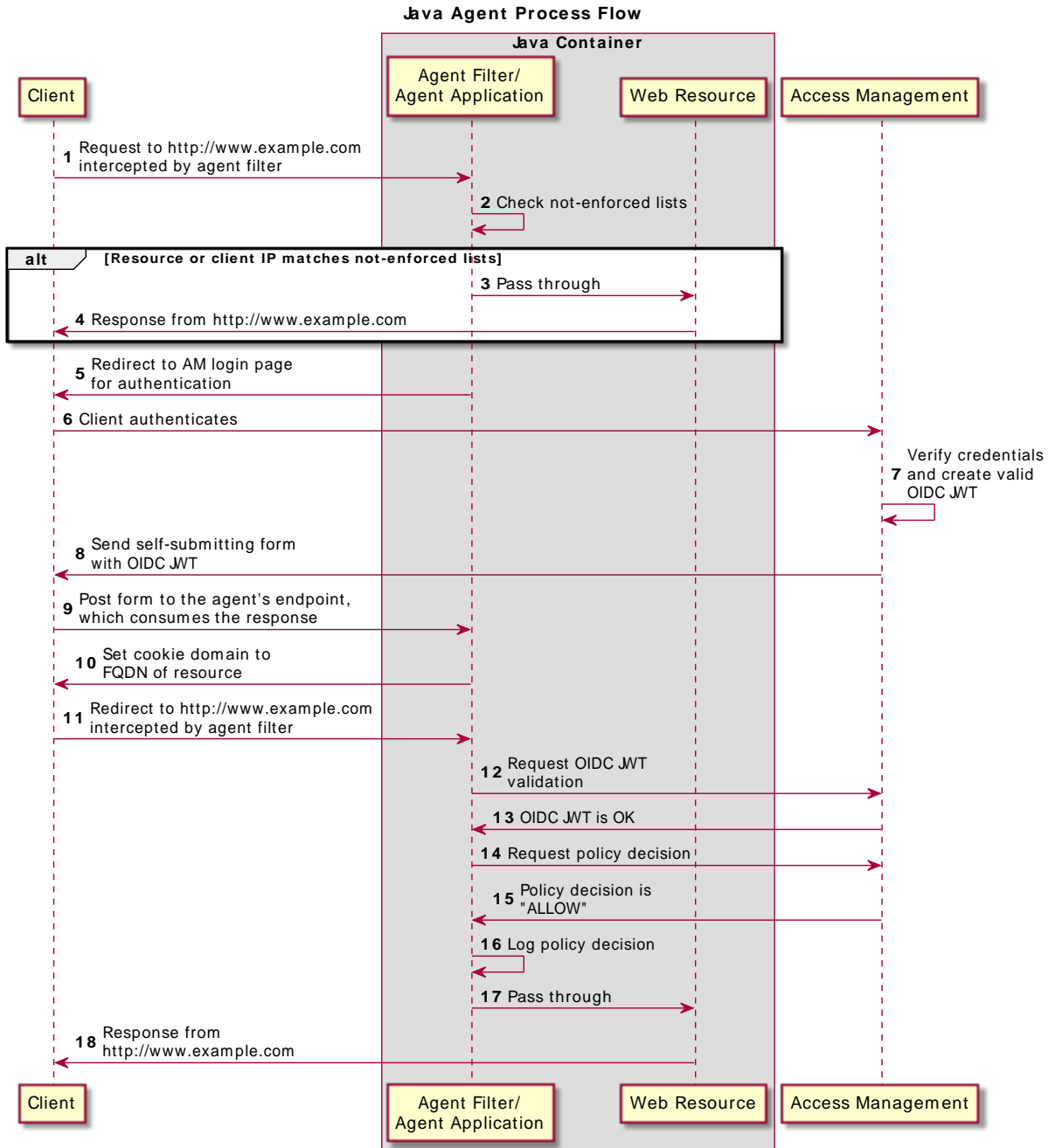
Suppose you wanted to withdraw money from your bank account using an ATM. The ATM would not allow you to access your account unless you identified yourself to the bank with your card and PIN number. For a joint account, you may also require additional authorization to access the funds.

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.

¹For a detailed diagram, see [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.

During client authentication, and to protect against reply attacks, the agent issues a *pre-authentication* cookie, whose name is configured in the Authentication Tracking Cookie Name property (`org.forgerock.agents.authn.cookie.name`). The agent uses this cookie to track the authentication request to AM.

Depending on the configuration, the agent may either issue a cookie to track all concurrent authentication requests, or may issue one cookie for each request.

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 where you can have more information about each one:

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

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.

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 more information about configuring continuous security properties, see [Continuous Security Properties](#).

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](#).

Autonomous "Fallback" Mode

You can start the Java agents in an autonomous "fallback" mode, which operates entirely independently, without having to contact an AM instance. In fallback mode, the agents allow access as defined in configured not-enforced lists; otherwise, access is denied.

In fallback mode, the agents evaluate not-enforced rules that use the following features:

- URLs, IP addresses, IP address ranges, and compound rules.
- Rules applied to specific HTTP methods.
- Inverted not-enforced rules, by using properties.
- Inverted not-enforced rules, by using inline logical operators.
- Rules that use regular expressions.
- Rules applied in the presence of named cookies with specified values.

Because the agent does not attempt to contact AM, the following functionality will not be available in fallback mode:

- Notifications
- Remote auditing
- Profile attributes
- Session attributes
- Response attributes
- Continuous security

To enable fallback mode, in the `/path/to/java_agents/agent_type/agent_instance/config/OpenSSOAgentBootstrap.properties` bootstrap properties file, set `org.forgerock.agents.fallback.mode.enabled=true`, and restart the Java container where the agent is installed.

Important

The agent must be configured for *Local configuration*, as there will be no attempt to contact AM when starting in fallback mode.

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 `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 `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 [Single Sign-On](#) and [Implementing CDSSO 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".

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 mode
- Custom Redirection Login Mode 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 set to `false`.**
- **Configure the following property:**
 - OAuth 2.0 Login List

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 OAuth 2.0 Login List 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 request².

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 `Allow Custom Login Mode`.

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.

²For more information, see the implementation details included in the [OpenAM Conditional Login URL](#) property.

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:

- **Ensure that the Allow Custom Login Mode 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:**
 - OpenAM Login URL (`com.sun.identity.agents.config.login.url`)
 - `org.forgerock.agents.legacy.login.url.list`

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 OpenAM Login URL property.

The agent appends a `resourceURL` parameter to the login endpoint.
 - If not configured, the agent matches the request with the domains and URLs specified by the `org.forgerock.agents.legacy.login.url.list` 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
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 <code>com.sun.identity.agents.polling.interval</code> 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
- Regular Expression Retain Query Parameters

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

- Remove Query Parameters

- Regular Expression Remove Query Parameters

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](#).

Authentication Failure Notification

By default, Java agents return an HTTP 400 message when they or AM are unable to authenticate the end user, regardless of the reason. This is to prevent malicious users from gaining information that could help them gain access to the system.

Consider an example where the agent returns an "unknown user" message. This would give malicious users a clue to keep on trying different user names until the error message changed to, for example, "wrong password".

The following is a table showing possible reasons for an agent to return an HTTP 400 message. Even if indirectly, they all mean that the agent cannot authenticate the end user:

Authentication Failure Reasons

Reason Code	Meaning
<code>AUTHN_BOOKKEEPING_COOKIE_MISSING</code>	<p>The agent cannot find the authentication tracking cookie (defined in the <code>org.forgerock.agents.authn.cookie.name</code> property).</p> <p>This error can happen if the user successfully authenticates, but clicks the back button of the browser to return to the previous page.</p>
<code>NONCE_MISSING</code>	<p>The agent found the authentication tracking cookie, but it cannot find the unique identifier of the authentication request inside the cookie.</p>

Reason Code	Meaning
	This error can happen if the user successfully authenticates, but clicks the back button of the browser to return to the previous page.
BAD_AUDIENCE	The audience in the JWT did not correspond to the audience in the cookie entry. This error can happen if all agents working in a cluster do not have the same Agent Profile Name.
NO_TOKEN	The agent cannot find the session ID token.
TOKEN_EXPIRED	The agent found the session ID token, but it is past its expiry date.
AM_SAYS_INVALID	The agent found the session ID token, the expiry time is correct, but AM returns that the ID token is invalid.
JWT_INVALID	The agent found the session ID token, but cannot parse it.
EXCEPTION	This reason can have the following meanings: <ul style="list-style-type: none"> • The agent found the session ID token, but threw an exception while parsing it. • The agent cannot connect to AM to validate the ID token, maybe due to a network outage.

The behavior of returning an HTTP 400 message is not always desirable; for example, when debugging the agent flow or when another application depends on the error message. To customize how the agent behaves in these cases, you can configure the following:

- A URL or URI where the agent redirects the end user after the authentication failure.
This way, you can control the message that the agent displays to the end user.
- User-friendly messages that are mapped to the different conditions that may cause the Java agent to return an HTTP 400 message.

For more information about the related properties, see [Authentication Failure 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 AM 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 AM Secret IDs for the Agents' OAuth 2.0 Provider in AM 6.0 or earlier"
- "To Configure AM Secret IDs for the Agents' OAuth 2.0 Provider in AM 6.5 or later"

To Configure AM 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 Security 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 AM 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 **rsajwt signingkey** 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 Default Mappings in the ForgeRock Access Management Security Guide*.

- c. Save your changes.

For more information about secret stores, see the chapter *Configuring Secret Stores of the ForgeRock Access Management Security 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. For example, **MyAgent**.

Agent URL

The URL where the Java agent resides, for example, **<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.

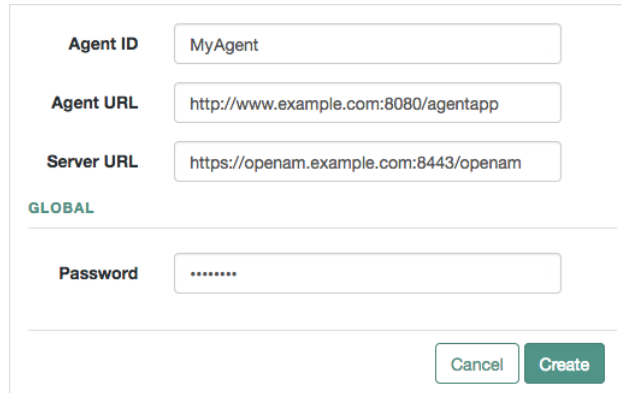
Server URL

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.



The screenshot shows a configuration form with the following fields and values:

- Agent ID: MyAgent
- Agent URL: http://www.example.com:8080/agentapp
- Server URL: https://openam.example.com:8443/openam
- GLOBAL (Section Header)
- Password:

At the bottom right of the form are two buttons: "Cancel" and "Create".

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 on the Global tab.
4. Adjust inheritance by toggling the Inherit value button next to properties that support inheritance:

🔒 = Do not inherit value from group.

🔓 = Inherit value from group.

Tip

You can also create agent profiles by using the `/realm-config/agents/WebAgent/{id}` endpoint in the REST API.

For more information, navigate to the API Explorer in your AM instance.

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.

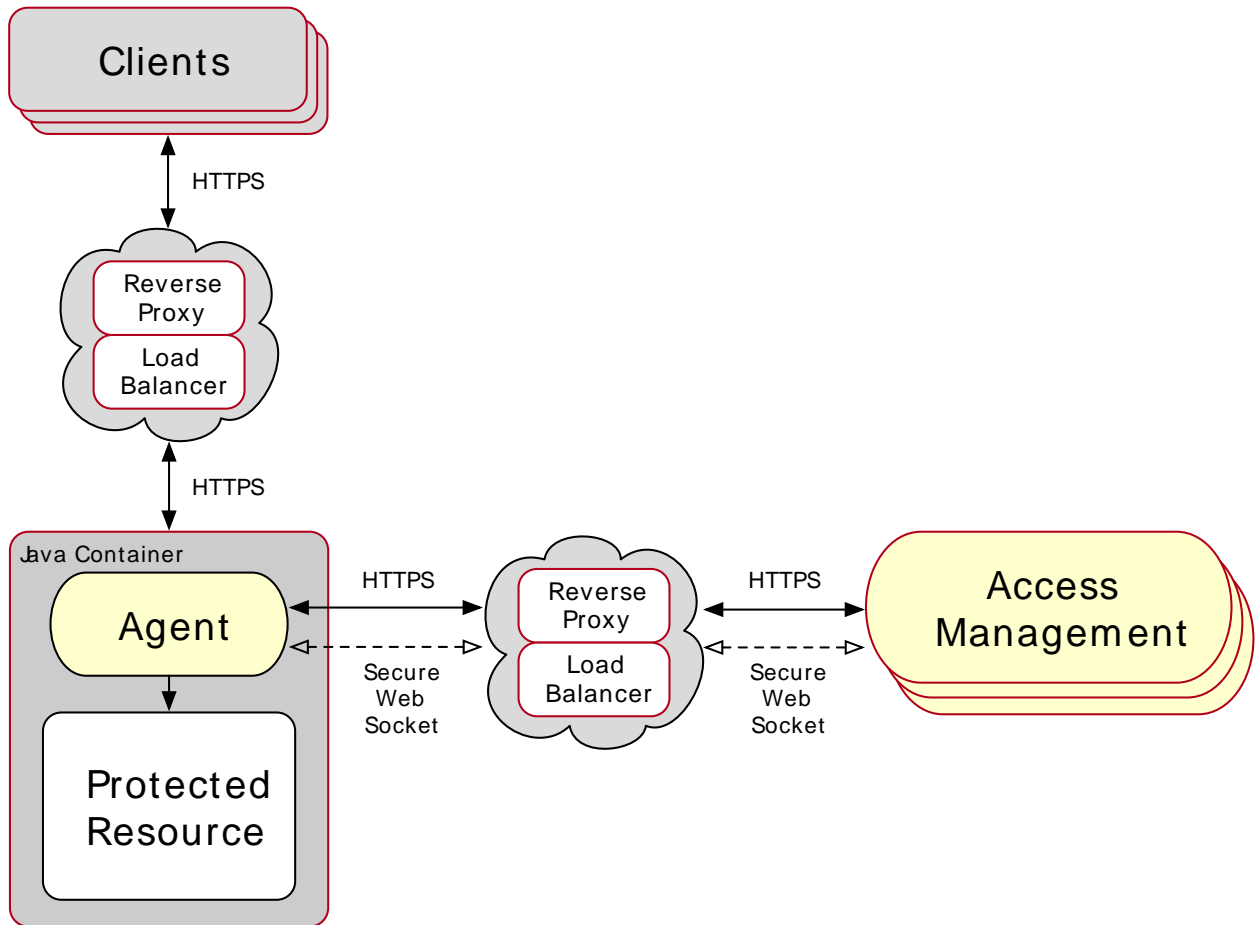
For more information, see "*Configuring Environments With Load Balancers and Reverse Proxies*".

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 Setup 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 AM 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 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.

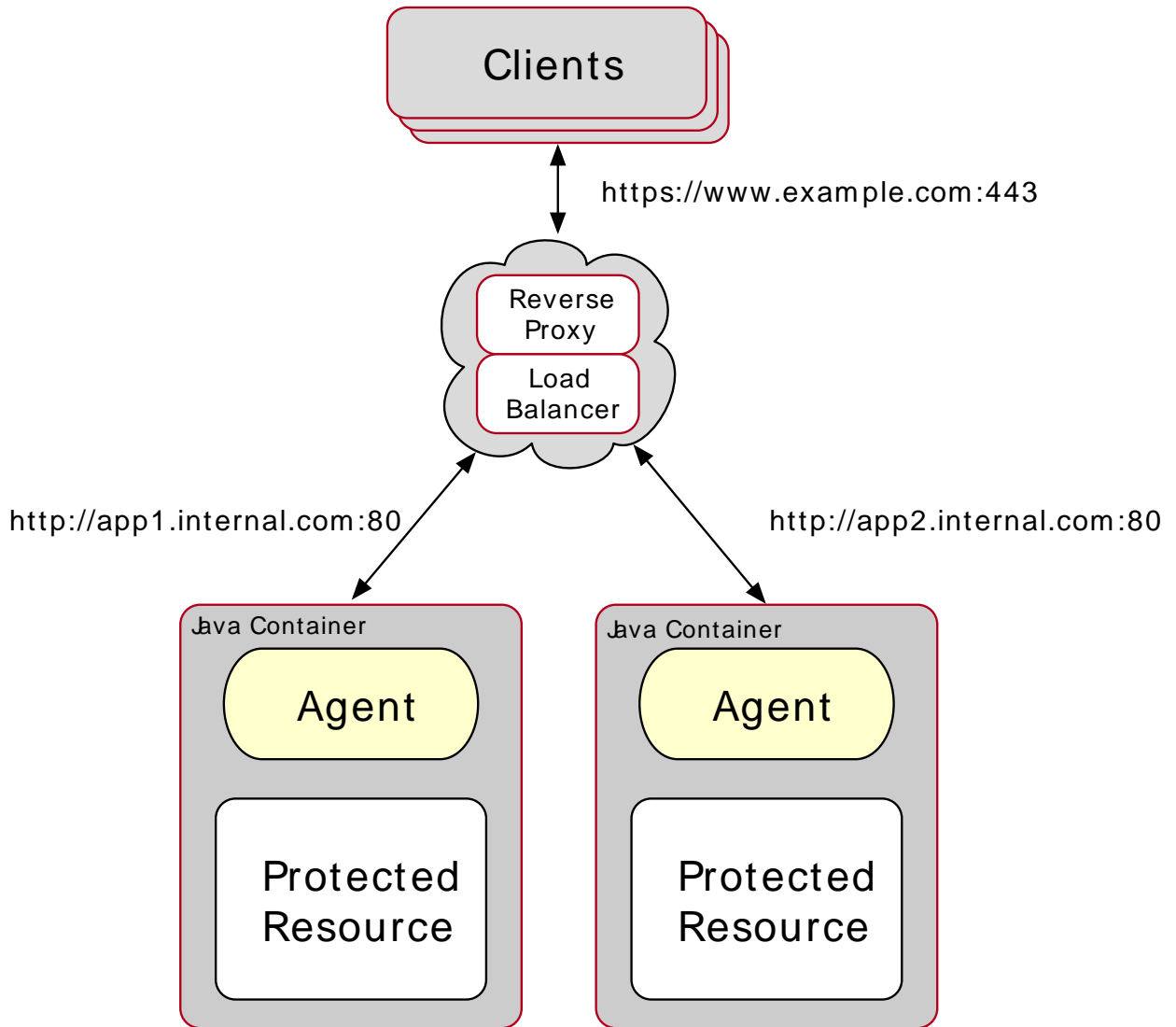
In this case, neither ports, protocols, or FQDNs match.

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:

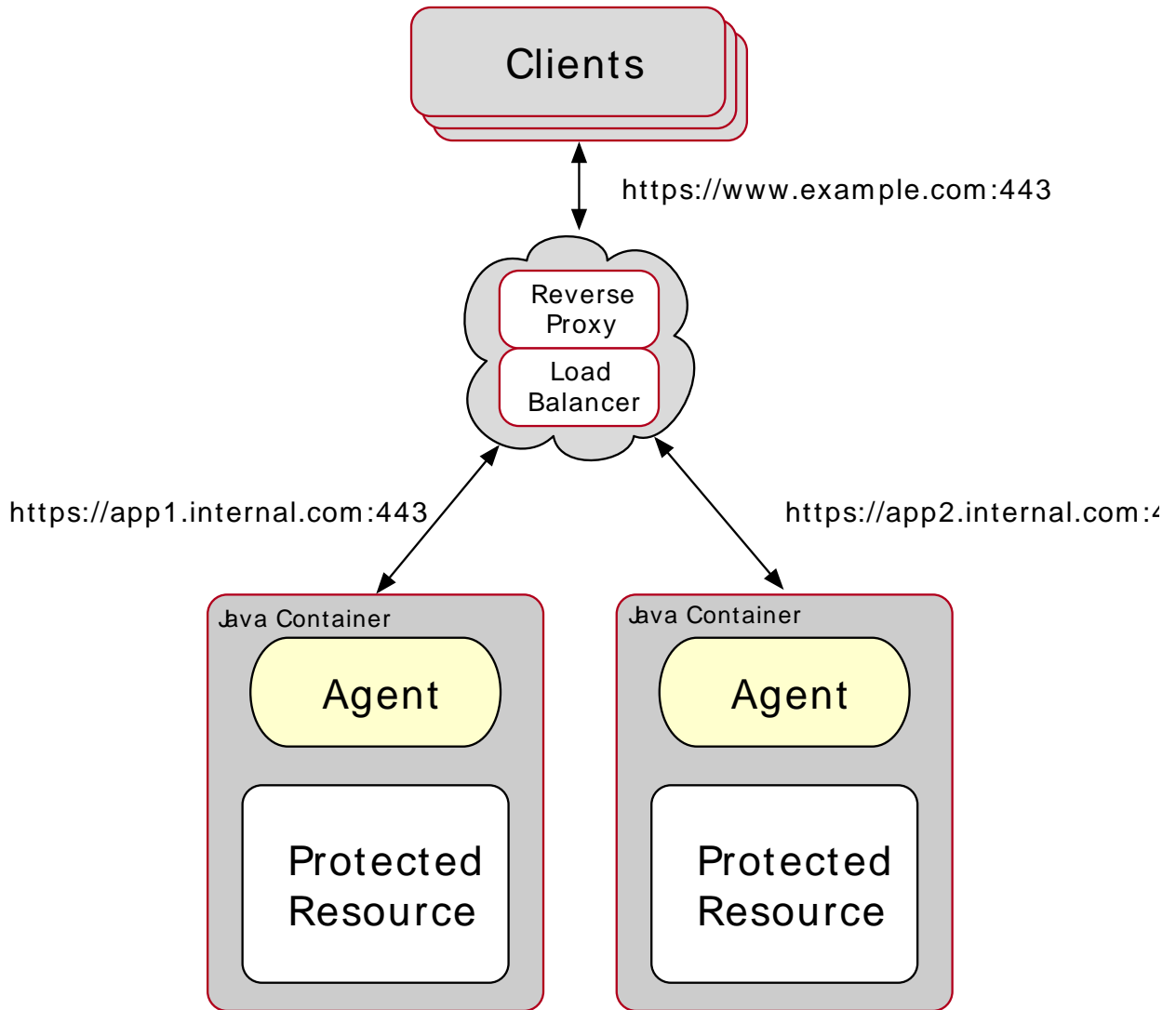
Different Protocol, Port, and FQDN



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:

Same Protocol and Port, Different FQDN



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`.

- (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`.

- 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

- Log in to the AM console with a user that has permissions to modify the Java agent profile.
- Navigate to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced.
- 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 `org.forgerock.agents.pdp.sticky.session.mode=[Cookie|URL]`.

- 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 `org.forgerock.agents.pdp.sticky.session.value=lb=myserver`.

- Save your changes.
- 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 5.7 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 *Configuring Policies*.
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 AM 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 *Enabling Restricted Tokens for CDSSO Session Cookies in the ForgeRock Access Management Security 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 `--acceptLicense` 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-notice/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/`

Directory for the debugging file. Use for troubleshooting.

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= /path/to/apache-tomcat/conf
AM_SERVER_URL= https://openam.example.com:8443/openam
CATALINA_HOME= /path/to/apache-tomcat
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 *Configuring Policies*.
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 AM 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 [Enabling Restricted Tokens for CDSSO Session Cookies in the ForgeRock Access Management Security 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 5.7 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 *Configuring Policies*.
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 AM 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 *Enabling Restricted Tokens for CDSSO Session Cookies* in the *ForgeRock Access Management Security 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 `--acceptLicense` 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-notice/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= /path/to/jboss
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 *Configuring Policies*.
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 AM 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 *Enabling Restricted Tokens for CDSSO Session Cookies in the ForgeRock Access Management Security 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

This section covers prerequisites and installation procedures for Java Agents 5.7 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 *Configuring Policies*.
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 AM 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 *Enabling Restricted Tokens for CDSSO Session Cookies* in the *ForgeRock Access Management Security 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 `--acceptLicense` 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-notice/license.txt`.
- b. Enter the absolute path to the root of the Jetty installation. For example, `/path/to/jetty/home`. This is the equivalent of the `JETTY_HOME` environment variable for Jetty.

```
This is the home of the Jetty installation (directory containing start.jar)
[ ? : Help, ! : Exit ]
Enter the Jetty home directory [/opt/jetty]: /path/to/jetty/home
```

- c. Enter the absolute path to the Jetty configuration directory. For example, `/path/to/jetty/etc`.

```
Enter the absolute path of the Jetty etc directory.
[ ? : Help, < : Back, ! : Exit ]
Enter the absolute path of the Jetty etc directory: /path/to/jetty/etc
```

- d. Enter the absolute path to the Jetty base directory. For example, `/path/to/jetty/base`. This is the equivalent of the `JETTY_BASE` environment variable for Jetty.

This path may be the same as the one specified as the root of the Jetty installation.

```
This is the base of the Jetty installation (directory containing the webapps subdirectory)
[ ? : Help, < : Back, ! : Exit ]
Enter the Jetty base directory [/usr/local/jetty]: /path/to/jetty/base
```

- 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/agentadmin
```

- g. 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
```

- 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 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 home directory (containing start.jar) : /path/to/jetty/home  
Jetty Server etc directory : /path/to/jetty/etc  
Jetty base directory (containing webapps subdirectory) which may be the same as your Jetty  
home directory : /path/to/jetty/base  
AM server URL : https://openam.example.com:8443/openam  
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/

Directory for the debugging file. Use for troubleshooting.

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 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= /path/to/jetty/etc
JETTY_HOME= /path/to/jetty/home
JETTY_BASE= /path/to/jetty/base
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 *Configuring Policies*.
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 AM 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 *Enabling Restricted Tokens for CDSSO Session Cookies in the ForgeRock Access Management Security 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 5.7 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 *Configuring Policies*.

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 AM 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 [Enabling Restricted Tokens for CDSSO Session Cookies in the ForgeRock Access Management Security 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 `--acceptlicense` 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-notice/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/bean_projects/domains/base_domain/startWebLogic.sh]: /path/to/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/bean/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= /path/to/Oracle_Home/user_projects/domains/base_domain/startWebLogic.sh
SERVER_NAME= AdminServer
WEBLOGIC_HOME_DIR= /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_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 *Configuring Policies*.
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 AM 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 *Enabling Restricted Tokens for CDSSO Session Cookies in the ForgeRock Access Management Security 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/opensscliendsdk.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 5.7 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 *Configuring Policies*.
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 AM 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 [Enabling Restricted Tokens for CDSSO Session Cookies in the ForgeRock Access Management Security 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 `--acceptlicense` 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>Node01/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:50080/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.

- Restart the WebSphere server.
- Deploy the `/path/to/java_agents/websphere_agent/etc/agentapp.war` agent application in WebSphere.
- To protect an application in the container, configure the agent filter.
- (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= /path/to/WebSphere/AppServer/profiles/AppSrv01/config/cells/DefaultCell01/nodes/
DefaultNode01/servers/server1
SERVER_INSTANCE_NAME= server1
HOME_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_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 *Configuring Policies*.
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 AM 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 *Enabling Restricted Tokens for CDSSO Session Cookies in the ForgeRock Access Management Security 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
<code>URL_POLICY</code>	Yes	Yes	AM performs the following tasks: <ul style="list-style-type: none"> 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
<code>SSO_ONLY</code>	Yes	No	AM issues an OIDC JWT to the client after successful authentication.
<code>NONE</code>	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.
<code>ALL</code>	Yes	Yes	This mode behaves in the same way as the <code>URL_POLICY</code> mode and is kept for backward-compatibility purposes.
<code>J2EE_POLICY</code>	-	-	<i>This mode does not apply to Java Agents 5.7, but it shows in the AM agent profile page for backward-compatibility purposes.</i>

^a For more information about AM authentication mechanisms, see the *ForgeRock Access Management Authentication and Single Sign-On Guide*.

^b For more information about AM policies, see the *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

- Navigate to Realms > Realm Name > Applications > Agents > Java > *Agent Name*.
- 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

Agent Filter Mode		
	SSO_POLICY	x
BankApp	URL_POLICY	x
Key	Value	+ Add

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 Security 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 Log Format](#) in the *ForgeRock Access Management Security 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 [Configuring the Trust Transaction Header System Property](#) in the *ForgeRock Access Management Security 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 property `Audit Types`, select the type of messages to log. For example, select `LOG_ALL` to log access allowed and access denied events.
3. In the property `Audit Log File Location`, 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 more information, 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 more information, 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 more information, 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 7 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 `Export Monitoring Metrics to CSV` property:
 - Set to `true` to configure the agent to write metric information to CSV files.
 - Set to `false` to prevent the agent from writing metric information to CSV files.

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

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. Configure AM to send cookies only when the communication channel is secure:

- a. In the AM console, select Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > SSO.
 - b. Enable Secure Cookies. The property `org.forgerock.agents.secure.cookies.enabled` is set to `true`.
2. 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.

3. 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_OPTS` 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"
```

4. 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 AM 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/boa/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/AppSrv01/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/AppSrv01/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 Security 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 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.

- "About Java Agent Properties"
- "Configuring Bootstrap Properties"
- "Configuring Global Properties"
- "Configuring Application Properties"
- "Configuring SSO Properties"
- "Configuring AM Services Properties"
- "Configuring Miscellaneous Properties"
- "Configuring Advanced Properties"

About Java Agent Properties

Property Aliases

A property alias specifies a path for a property. One property can have an unlimited number of aliases, however, an alias must be unique.

¹ See "Configuration" 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.

Aliases starting with `org.forgerock` follow a naming convention to provide information about the property.

When a property has multiple aliases, the agent evaluates the aliases in alphabetical order. If the aliases each specify a different value for the property, the agent assigns the value specified by the first alias in the alphabetical order, and then propagates that value to the other aliases.

The following example assigns different values to a property with three aliases:

```
com.sun.identity.agents.app.username=AGENT3
com.sun.identity.agents.config.profilename=AGENT1
org.forgerock.agents.profile.name=AGENT2
```

The agent evaluates `com.sun.identity.agents.app.username` first, and propagates that value to the other aliases, resulting in this:

```
com.sun.identity.agents.app.username=AGENT3
com.sun.identity.agents.config.profilename=AGENT3
org.forgerock.agents.profile.name=AGENT3
```

Property Types

Properties can have the following types:

- String (default)
- Enumeration
- List of strings
- Map of strings
- Integer
- Boolean string, where the string `true` returns `true`, and all other strings return `false`.

If the value assigned to a property does not conform to the property type, the default value of the property is assigned and an error is written to the logs.

Configuring Bootstrap Properties

Bootstrap properties are set in the `config/0openSSOAgentBootstrap.properties` file.

`am.encrypted.pwd`

When using an encrypted password, set this to the encryption key used to encrypt the agent profile password.

`org.forgerock.agents.fallback.mode.enabled`

When `true`, the Agent runs in autonomous, "fallback" mode, independently of an AM instance.

The Agents allow access using only not-enforced rules. Access to any resource not covered by not-enforced rules is denied. For more information, see "Autonomous "Fallback" Mode".

Default: `false`

Type: Boolean

Hot-swap: No

Bootstrap property: Yes

Property: `org.forgerock.agents.fallback.mode.enabled`, introduced in Java Agent 5.7

Alias: `com.forgerock.agents.config.fallback.mode`, introduced in Java Agent 5.7

`com.ipplanet.am.naming.url`

⚠ *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

`org.forgerock.agents.encrypted.password`

When using a plain text password, set this to the password for the agent profile, and leave `am.encrypted.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: Empty

Type: String

Hot-swap: No

Bootstrap property: Yes

Property: `org.forgerock.agents.encrypted.password`, introduced in Java Agent 5.5.2

Alias: `com.ipplanet.am.service.secret`, introduced in Java Agent 5.5.2

`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 AM Host, AM Port, AM Protocol, and AM Path.

`org.forgerock.agents.am.path`

The URI under which AM is deployed, such as `/openam`.

Default: Empty

`org.forgerock.agents.csv.monitoring.directory`

The full path to the directory where the Java agent writes debug log files and CSV files. See Debug Logfile Directory.

Default: None

`org.forgerock.agents.local.audit.file.path`

The full path of the Java agent's local audit log file. See Audit Logfile Path.

Default: None; local auditing is disabled

`org.forgerock.agents.lock.config.enabled`

When `true`, specifies that an agent restart is required to allow agent configuration changes, even for hot-swappable parameters.

Default: `false`

Type: Boolean

Bootstrap property: Yes

Hot-swap: No

Property: `org.forgerock.agents.lock.config.enabled`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.lock.enable`, introduced in Java Agent 5.5.2

`org.forgerock.agents.agent.profile.realm`

The realm name where the agent authenticates to AM.

Default: `/` (top-level realm)

`org.forgerock.agents.profile.name`

The profile name used to fetch agent configuration data from AM.

Default: Empty

Type: String

Hot-swap: No

Bootstrap property: Yes

Property:

- `org.forgerock.agents.profile.name`, introduced in Java Agent 5.5.2
- `com.sun.identity.agents.app.username`, introduced in Java Agent 5.5.2
- `com.sun.identity.agents.config.profilename`, introduced in Java Agent 5.5.2

`com.sun.identity.agents.config.service.resolver`

⚠ This property does not apply to Java Agents 5.7, although it might appear in the AM console.

Service Resolver Class Name

+ Not available in the console for AM 6.5.x and earlier versions.

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The name of a service resolver class to override the ForgeRock service resolver, and instantiate a new service resolver. Use this property to customize pre-handlers and post-handlers.

For information about how to add a custom task handler to the list of handlers the agent runs for each resource request, and some example handlers that you can use, see "Implementing Custom Task Handlers".

At startup, the Agent tries to instantiate the specified service resolver class. If unsuccessful, it instantiates the original service resolver.

Use the following functions to return a list of class names to customize the task handler:

- `List<String> getPreInboundTaskHandlers()`: Nominated classes are executed *before* all other inbound task handlers, and must implement `IAMFilterTaskHandler`.
- `List<String> getPostInboundTaskHandlers()`: Nominated classes are executed *after* all other inbound task handlers, and must implement `IAMFilterTaskHandler`.
- `List<String> getPreSelfRedirectHandlers()`: Nominated classes are executed *before* all other self-redirect task handlers, and must implement `IAMFilterTaskHandler`.
- `List<String> getPostSelfRedirectHandlers()`: Nominated classes are executed *after* all other self-redirect task handlers, and must implement `IAMFilterTaskHandler`.

- `List<String> getPreFilterResultHandlers()`: Nominated classes are executed *before* all other result handlers, and must implement `IAMFilterResultHandler`.
- `List<String> getPostFilterResultHandlers()`: Nominated classes are executed *after* all other result handlers, and must implement `IAMFilterResultHandler`.

If the named handler classes are not on the classpath, or do not implement the required interface, then:

- Handler instantiation fails.
- A message is logged at ERROR level.
- The agent abandons processing and returns an HTTP 500, effectively denying all requests.

When a handler list is built, make sure that your function returns `true` if appropriate. Handlers whose `isActive` is `false` are evicted from the handler list.

The `process` function for inbound and self-redirect handlers is invoked until a non-null value, such as `continue` or `block`, is returned. The non-null value becomes the result for that resource access. Returning null indicates to carry on to the other handlers. Any filter result handler that returns a null value causes an error.

Manually edit in bootstrap properties: Yes

Default: `ServiceResolver.class name`

Type: String

Hot-swap: No

Property: `org.forgerock.agents.service.resolver.class.name`, introduced in Java Agent 5.6.2.1

Configuring Global Properties

This section describes 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.

- "Profile Properties"
- "General Properties"
- "Debug and Metric Properties"
- "Audit Properties"
- "User Mapping Properties"
- "Fully Qualified Domain Name Checking Properties"

Profile Properties

Group

Assign the Java agent to a previously configured group in order to inherit selected properties from the group.

Password

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 `org.forgerock.agents.encrypted.password`. For more information, see "Configuring Bootstrap Properties".

Status

The status of the agent configuration.

Agent Profile Realm

The realm name where the agent authenticates to AM.

Default: / (top-level realm)

Type: String

Hot-swap: No

Property: `org.forgerock.agents.agent.profile.realm`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.organization.name`, introduced in Java Agent 5.5.2

Configuration Enabled

When `true`, specifies that an agent restart is required to allow agent configuration changes, even for hot-swappable parameters.

Default: False

Type: Boolean

Bootstrap property: Yes

Hot-swap: No

Property: `com.sun.identity.agents.config.lock.enable`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.organization.name`, introduced in Java Agent 5.5.2

Configuration Repository

The location of the Java agent configuration:

- **LOCAL**: Configuration is read from `OpenSSOAgentConfiguration.properties` only.
Add the following property to the bootstrap file: `com.sun.identity.agents.config.repository.location=local`
- **REMOTE**: Configuration is downloaded from AM.
- **CENTRALIZED**: This deprecated value will be substituted for **REMOTE**.

For more information, see "Configuration".

Note

At startup, the agent reads the bootstrap properties file, and then the configuration properties file.³

If this property is set to **LOCAL**, the agent will use all of the properties it has retrieved and continue working.

If this 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: **REMOTE**

Type: Constrained string

Requires restart if changed: Yes

Property: `org.forgerock.agents.config.location`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.repository.location`, introduced in Java Agent 5.5.2

Configuration Reload Interval

The time interval in seconds after which the Java agent reloads the agent profile if it has been changed since it was last read. The agent configuration location is given in Configuration Repository.

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.

Default: **0**; disabled

Type: Integer

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.

Hot-swap: Yes

Property: `org.forgerock.agents.config.reload.seconds`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.load.interval`, introduced in Java Agent 5.5.2

Fall-Forward Mode

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

Determines how the Agent behaves when AM is unavailable, and when `org.forgerock.agents.fallback.mode.enabled` is NOT enabled (default behavior):

- `false`: The Agent denies every incoming request for a resource, with an HTTP 403, even if the resource matched a *not enforced* rule at the time that AM became unavailable.
- `true`: The Agent allows every incoming request for a resource matched by a *not enforced* rule, until AM is available, and a config change notification tells the agent to reload the *not enforced* rules.

Default: `false`

Type: Boolean

Hot-swap: Yes

Bootstrap property: No

Property: `org.forgerock.agents.fallforward.mode.enabled`, introduced in Java Agent 5.7

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 cookie name must be unique across the set of cookies the user's browser receives, because some browsers behave in unexpected ways if they receive several cookies with the same name. For example, do not set the JWT cookie name to `iPlanetDirectoryPro`, which is the default name of AM's session cookie.

Default: `am-auth-jwt`

Type: String

Hot-swap: Yes

Property: `org.forgerock.agents.jwt.cookie.name`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.openam.agents.config.jwt.name`, introduced in Java Agent 5.5.2

Convert SSO Tokens into OpenID Connect JWTs

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

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 to allow users to 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 negates the need for 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 causes the agent to request additional JWTs from AM.

Default: `false`

Type: Boolean

Hot-swap: Yes

Property: `org.forgerock.agents.accept.ipdp.cookie.enabled`, introduced in Java Agent 5.7

Alias: `com.forgerock.agents.accept.ipdp.cookie`, introduced in Java Agent 5.6.2.1

JWT Cookie Domain List

🔔 *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

A list of domains in which the Agent attempts to create JWT cookies:

- If the list is empty, the agent creates cookies only in its own domain.
- If the agent is running behind a browser, it can create cookies only in its own domain.
- If the agent is running behind a proxy, it should be able to create cookies in any required domains.

Default: Empty

Type: List of strings

Hot-swap: Yes

Bootstrap property: Yes

Property: `org.forgerock.agents.jwt.cookie.domain.list`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.cdsso.domain`, introduced in Java Agent 5.5.2

Secure Cookies

🔗 *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

When `true`, the agent marks cookies as secure, sending them only if the communication channel is secure. Set to `true` when agent connections are over SSL.

Default: `false`

Type: Boolean

Hot-swap: Yes

Property: `org.forgerock.agents.jwt.cookie.secure.enabled`, introduced in Java Agent 5.6.2.1

Balance WebSocket Connection Interval Minutes

The time in minutes before WebSockets to AM are killed and reopened. This property helps ensure a balanced distribution of connections across the AM servers on the site.

Default: `30`

Type: Integer

Hot-swap: Yes

Property: `org.forgerock.agents.balance.websocket.interval.minutes`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.openam.agents.config.balance.websocket.connection.interval.in.minutes`, introduced in Java Agent 5.5.2

JWT Cache Size

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The maximum number of entries in the JWT cache.

Default: 5000

Type: Integer

Hot-swap: No

Property: `org.forgerock.agents.jwt.cache.size`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.openam.agents.config.jwt.cache.size`, introduced in Java Agent 5.5.2

JWT Cache TTL

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The time in minutes after which entries in the JWT cache timeout and are purged.

Parsing JWTs is a CPU intensive process. Because all JWTs in the cache have already been parsed, it is advantageous to configure a long timeout for this cache.

Default: 30

Type: Integer

Hot swap: No

Property: `org.forgerock.agents.jwt.cache.ttl.minutes`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.openam.agents.config.jwt.cache.ttl.minutes`, introduced in Java Agent 5.5.2

Exchanged SSO Token Cache Time to Live

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The time in minutes after which entries in the SSO token exchange cache timeout and are purged.

The returned JWT is cached against the relevant SSO token. If the same SSO token is presented again, 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.

Because exchanging SSO tokens for JWTs is an expensive process, previously exchanged SSO tokens are cached. When an entity is unable to permanently store its JWT in a cookie, calls to AM can be avoided.

Default: 5

Type: Integer

Hot swap: No

Property: `org.forgerock.agents.sso.exchange.cache.ttl.minutes`, introduced in Java Agent 5.6.2.1

Exchanged SSO Token Cache Size

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The maximum number of entries in the SSO Exchange cache, used in the exchange of an SSO tokens for a JWT.

When the maximum is reached, the oldest records are overwritten.

Default: 100

Type: Integer

Hot swap: No

Property: `org.forgerock.agents.sso.exchange.cache.size`, introduced in Java Agent 5.6.2.1

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

Idle Time Refresh Windows

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The time interval, in minutes, the agent waits before calling 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

Type: Integer

Hot-swap: yes

Property: `org.forgerock.agents.idle.time.window.minutes`, introduced in Java Agent 5.6.2.1

Internal Monitoring URI

A dummy endpoint that when invoked produces information about the current Agent properties.

Default: `/agentapp/propertyInfo`

Type: URI string

Hot-swap: Yes

Property: `org.forgerock.agents.property.info.endpoint.uri`, introduced in Java Agent 5.5.2

HTTP Session Binding

When `true`, 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`

Type: Boolean

Hot-swap: Yes

Property: `org.forgerock.agents.http.session.binding.enabled`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.httpsession.binding`, introduced in Java Agent 5.5.2

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` (disabled)

Type: Integer

Hot-swap: Yes

Property: `org.forgerock.agents.login.attempt.limit.count`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.login.attempt.limit`, introduced in Java Agent 5.5.2

Login Attempt Limit Cookie Name

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The name of the cookie used to record the number of login attempts.

Default: `amFilterParam`

Type: String

Hot-swap: Yes

Property: `org.forgerock.agents.login.counter.cookie.name`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.login.counter.name`, introduced in Java Agent 5.5.2

Custom Response Header

Custom headers the Java agent sets for the client. The key is the header name. The value is the header value. For example, `org.forgerock.agents.response.header.map[Cache-Control]=no-cache`.

Default: Empty

Type: Map, with format *from header name:to header name*

Hot-swap: Yes

Property: `org.forgerock.agents.response.header.map[HEADER_NAME]=HEADER_VALUE`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.response.header`, introduced in Java Agent 5.5.2

Redirect Attempt Cookie Name

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

Detect redirect loops while authenticating, which can indicate a cookie domain problem. The Agent does this by using a cookie to hold the current redirection count.

Default: `amFilterRDParam`

Type: String

Hot-swap: Yes

Property: `org.forgerock.agents.redirect.cookie.name`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.redirect.cookie.name`, introduced in Java Agent 5.5.2

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`

Type: Integer

Hot-swap: Yes

Property: `org.forgerock.agents.redirect.attempt.limit`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.redirect.attempt.limit`, introduced in Java Agent 5.5.2

Debug and Metric Properties

Debug Level

Debugging messages are logged at the following levels.

- **OFF**: No logging is performed, except for critical errors which cannot be disabled.
- **ERROR**: Only errors are logged.
- **WARNING**: Errors and warnings are logged.
- **MESSAGE**: Errors, warnings and informative messages are logged.
- **TRACE**: Highest level of logging available.
- **ON**: Highest level of logging that is written to the standard output, as opposed to file.

Note that not all containers are good at capturing messages logged to the standard output, and warnings or critical errors can easily disappear forever.

Default: **ERROR**

Type: Constrained string

Property: `org.forgerock.agents.debug.level`, introduced in Java Agent 5.5.2

Alias: `com.ipianet.services.debug.level`, introduced in Java Agent 5.5.2

Debug File Rotation Prefix

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

A prefix to append to the start of the log file name.

Default: Empty

Type: String

Property: `org.forgerock.agents.debug.prefix`, introduced in Java Agent 5.7

Alias: `org.forgerock.openam.debug.prefix`, introduced in Java Agent 5.7

Debug File Rotation Suffix

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

A suffix to append to the end of the log file name when it is rotated. For example, `-yyyy.MM.dd-HH.mm`. Note that file rotation can be disabled.

The suffix can be defined freely, however, if it does not include a timestamp that produces different file names when the rotation time is reached, log file rotation can fail. Invalid values produce exceptions in the container logs or agent logs.

Default: Empty. If log rotation is disabled, nothing is appended to the log file name. If log rotation is enabled and this property is not set, the value `-yyyy.MM.dd-HH.mm` is used.

Type: String. For information about how to configure this property, see the `java.text.SimpleDateFormat` Java class in the Java SDK's documentation.

Property: `org.forgerock.agents.debug.suffix`, introduced in Java Agent 5.7

Alias: `org.forgerock.openam.debug.suffix`, introduced in Java Agent 5.7

Debug File Rotation Size

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The approximate size in bytes at which a log file is rotated to a new log file.

If this property is set, and no value is assigned to Debug File Rotation Suffix, the suffix `-yyyy.MM.dd-HH.mm` is used.

Default: Not set; file rotation is disabled

Type: Integer

Property: `org.forgerock.agents.debug.rotation.size.bytes`, introduced in Java Agent 5.7

Alias: `org.forgerock.openam.debug.rotation.maxsize`, introduced in Java Agent 5.7

Debug Log File Rotation Time

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The time in minutes, after which a log file is rotated to a new log file.

If this property is set, and no value is assigned to Debug File Rotation Suffix, the suffix `-yyyy.MM.dd-HH.mm` is used.

Default: Not set; file rotation is disabled

Type: Integer

Property: `org.forgerock.agents.debug.rotation.time.minutes`, introduced in Java Agent 5.7

Alias: `org.forgerock.openam.debug.rotation`, introduced in Java Agent 5.7

Debug File Rotation Retention Count

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The number of log files to retain after rotation. For example, specifying `10` would give you a current debug file, and nine older, rotated files.

When the specified limit is reached, the oldest log file is deleted when a file rotation occurs.

Default: None; all rotated log files are kept

Type: Integer

Property: `org.forgerock.agents.debug.retention.count`, introduced in Java Agent 5.7

Debug Logfile Directory

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The full path to the directory where the Java agent writes debug log files and CSV files.

Default: None; no logging occurs until a value for this property is provided. Anything logged is written to the standard output and possibly the container log file.

Type: String

Hot swap: No

Bootstrap property: Yes

Property: `org.forgerock.agents.csv.monitoring.directory`, introduced in Java Agent 5.7

Alias: `com.ipplanet.services.debug.directory`, introduced in Java Agent 5.5.2

Export Monitoring Metrics to CSV

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

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`

Type: Boolean

Hot-swap: Yes

Property: `org.forgerock.agents.monitoring.to.csv.enabled`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.agents.config.monitoring.to.csv`, introduced in Java Agent 5.5.2

Audit Properties

Audit Types

The type of messages to audit. Valid values include:

- `LOG_NONE`: Disable audit logging.
- `LOG_ALLOW`: Audit allowed requests
- `LOG_DENY`: Audit denied requests.
- `LOG_BOTH`: Audit allowed and denied requests.

Default: `LOG_NONE`

Type: Constrained string

Hot-swap: Yes

Property: `org.forgerock.agents.audit.what`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.audit.accesstype`, introduced in Java Agent 5.5.2

Audit Log File Location

The location where the Java agent logs audit messages. If `Audit Types` is `LOG_NONE`, this property has no effect. Valid values include:

- `NONE`: Disable audit logging.
- `LOCAL`: Log audit event messages locally to the agent installation.

- **REMOTE**. Log audit event messages to the audit event handler configured in the AM realm.
- **ALL**. Log audit event messages locally and to the audit event handler configured in the AM realm.

Default: **NONE**

Type: Constrained string

Hot-swap: yes

Property: `org.forgerock.agents.audit.where`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.log.disposition`, introduced in Java Agent 5.5.2

Audit Log Rotation Enabled

When true, rotate local audit log files that have reached the size specified by the Local Audit Log File Rotation Size property.

Default: **False**

Hot-swap: yes

Property: `org.forgerock.agents.local.audit.log.rotation.enabled`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.local.log.rotate`, introduced in Java Agent 5.5.2

Local Audit Log File Rotation Size

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**

Type: Integer

Hot-swap: yes

Property: `org.forgerock.agents.local.audit.log.rotation.bytes`, introduced in Java Agent 5.7

Alias: `com.sun.identity.agents.config.local.log.size`, introduced in Java Agent 5.5.2

Audit Logfile Retention Count

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The number of audit log files to retain after rotation. For example, specifying **10** would give you a current log file, and nine older, rotated files.

When the specified limit is reached, the oldest log file is deleted when a file rotation occurs.

Default: All rotated log files are kept

Type: Integer

Property: `org.forgerock.agents.local.audit.log.retention.count`, introduced in Java Agent 5.7

Audit Logfile Path

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The full path of the Java agent's local audit log file.

Default: None; local auditing is disabled

Type: String

Hot swap: No

Bootstrap property: Yes

Property: `org.forgerock.agents.local.audit.file.path`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.local.logfile`, introduced in Java Agent 5.5.2

Remote Log File Name

🔗 *This property does not apply to Java Agents 5.7, although it may appear in the AM console.*

Property: `com.sun.identity.agents.config.remote.logfile`

User Mapping Properties

User Mapping Mode

Specifies where to obtain the user ID, as follows:

- `USER_ID`: The agent reads the property User DN Flag:
 - If `true`, the user ID is set to the principal user name.
 - If `false`, the user ID is set to the value of the session property specified by User Session Property Name.
- `PROFILE_ATTRIBUTE`. The user ID is set to the value of a named profile attribute specified by User Attribute Name.

- **HTTP_HEADER**. The user ID is set to the value of a named HTTP header specified by User Attribute Name.
- **SESSION_PROPERTY**. The user ID is set to the value of a named session property specified by User Attribute Name.

If the user ID cannot be set, the user is not logged in and access requests are denied.

Default: `USER_ID`

Type: Constrained string

Property: `org.forgerock.agents.user.mapping.mode`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.user.mapping.mode`, introduced in Java Agent 5.5.2

User Attribute Name

When the User Mapping Mode is **HTTP_HEADER**, this property is the name of the HTTP header attribute to identify the user. The named header must be present in the incoming headers.

Default: `employeenumber`

Type: String

Property: `org.forgerock.agents.user.mapping.mode.attribute.name`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.user.attribute.name`, introduced in Java Agent 5.5.2

User DN Flag

When User Mapping Mode is **USER_ID**, this flag indicates whether to identify the user through the user DN, as follows:

- If **true**, the DN is taken from `universalId`, retrieved from the AM user session info.
- If **false**, the user is identified by the value of `USER_ID_MODE_SESSION_PROPERTY_NAME`.

Default: `false`

Type: Boolean

Property: `org.forgerock.agents.userid.mapping.mode.use.dn.enabled`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.user.principal`, introduced in Java Agent 5.5.2

User Session Property Name

When User Mapping Mode is **USER_ID** and User DN Flag is **false**, the user is identified by the value of this property.

Default: `UserToken`

Type: String

Property: `org.forgerock.agents.userid.mapping.mode.use.session.property.name`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.user.token`, introduced in Java Agent 5.5.2

Fully Qualified Domain Name Checking Properties

FQDN Check

Enables checking of the FQDN default value and FQDN map value.

Default: `false`

Type: Boolean

Property: `org.forgerock.agents.fqdn.check.enabled`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.fqdn.check.enable`, introduced in Java Agent 5.5.2

FQDN Default

The default FQDN to use for the incoming server, if the agent cannot find a value in the FQDN map. If this property is not defined, FQDN checking is disabled.

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 an FQDN to this property, you must also add it to the Agent Root URL for CDSSO property.

Default: Empty

Type: String

Property: `org.forgerock.agents.fqdn.default`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.fqdn.default`, introduced in Java Agent 5.5.2

FQDN Map

Maps invalid server names to valid server names. Map keys are case-insensitive.

Default: Empty

Type: Map, with format `invalid server name:valid server name`

Property: `org.forgerock.agents.fqdn.map`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.fqdn.mapping`, introduced in Java Agent 5.5.2

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
- Response Attributes
- Common Attributes Fetching Processing Properties
- Session Attributes
- Privilege Attributes Processing Properties
- Custom Authentication Processing Properties
- Continuous Security Properties
- Query Parameter Handling Properties
- Authentication Failure 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: Empty

Property: `com.sun.identity.agents.config.login.form`

Login Error URI

ⓘ This property does not apply to Java Agents 5.7, although it might appear in the AM console.

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: Empty

Property: `com.sun.identity.agents.config.login.error.uri`

Use Internal Login

ⓘ This property does not apply to Java Agents 5.7, although it might appear in the AM console.

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

ⓘ This property does not apply to Java Agents 5.7, although it might appear in the AM console.

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`

Redirect to AM's Success URL

+ Not available in the console for AM 6.5.x and earlier versions.

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

- `true`: The agent redirects to the success URL specified in the AM service, if any.
If no success URL is specified in AM, the agent redirects to the original requested URL, if any.
- `false`: The agent redirects to the requested URL, if any.

For more information about configuring success URLs in AM, see *Configuring Success and Failure Redirection URLs*.

Default: `false`

Type: Boolean

Hot-swap: Yes

Property: `org.forgerock.agents.authn.success.redirect.session.url.enabled`, introduced in Java Agent 5.6.3

Authentication Exchange URI

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

Allows the administrator to define an endpoint to facilitate the exchange of SSO tokens for OIDC JWTs.

Default: Empty

Type: String (URL of a dummy endpoint within the Agent)

Hot-swap: Yes

Property: `org.forgerock.agents.authn.exchange.uri`, introduced in Java Agent 5.7

Authentication Exchange Cookie Name

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

Allows the administrator to define an endpoint to facilitate the exchange of SSO tokens for OIDC JWTs.

Default: Empty

Type: String (valid cookie name)

Hot-swap: Yes

Property: `org.forgerock.agents.authn.exchange.cookie.name`, introduced in Java Agent 5.7

Logout Processing Properties

Application Logout Handler

🔔 *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

Property: `com.sun.identity.agents.config.logout.application.handler`

Application Logout URI

A map of request URIs that cause logout of the user session when invoked. Use the `key:value` format *web application name:logout URI*.

To set a global logout URI for applications without other logout URIs defined, leave the key empty, and set the value as `/logout.jsp`.

To set a logout URI for a specific application, set the key to the name of the application, and set the value to the value of application logout page.

Default: Empty

Type: Map, with format *application name:logout URI*

Hot-swap: Yes

Property: `org.forgerock.agents.logout.endpoint.map`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.logout.uri`, introduced in Java Agent 5.5.2

Logout Request Parameter Map

Map of parameters in the HTTP request that trigger logout events. Use the `key:value` format *application name:parameter name to trigger logout*.

To set a global logout request parameter for applications without other logout request parameters defined, leave the key empty, and set the value to `logoutparam`.

To set a logout URI for a specific application, set the key to the name of the application, and set the value to an application logout request parameter, such as `logoutparam`.

Default: Empty

Type: Map, with format *application name:HTTP parameter name*

Hot-swap: Yes

Property: `org.forgerock.agents.logout.request.param.map`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.logout.request.param`, introduced in Java Agent 5.5.2

Logout Introspect Enabled

When `true`, the Java agent checks the HTTP request body to locate the value of Logout Request Parameter Map.

Default: `false`

Type: Boolean

Hot-swap: Yes

Property: `org.forgerock.agents.logout.introspection.enabled`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.logout.introspect.enabled`, introduced in Java Agent 5.5.2

Logout Goto URI

A map of request URIs to go to after logout using an endpoint defined in Application Logout URI.

To set a global URI for applications without other logout URIs defined, leave the key empty, and set a return URI such as `/return.html`.

To set a logout URI for a specific application, set the key to the name of the application, and set the return URI to the value of application logout entry URI, such as `/myApp/return.html`.

Default: Empty

Type: Map, with format *application name:goto URI*

Hot-swap: Yes

Property: `org.forgerock.agents.logout.goto.map`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.logout.entry.uri`, introduced in Java Agent 5.5.2

Access Denied URI Processing Properties

Resource Access Denied URI

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`.

Tip

Specify a full URL if required, including the host name. For example: `https://help.example.com/errors/accessdenied.html`.

Default: Empty

Type: Map, with format *application name:URI*

Hot-swap: Yes

Property: `org.forgerock.agents.access.denied.uri.map`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.access.denied.uri`, introduced in Java Agent 5.5.2

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: `|` (pipe)

Type: String

Hot-swap: Yes

Property: `org.forgerock.agents.notenforced.compound.separator`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.notenforced.rule.compound.separator`, introduced in Java Agent 5.5.2

Not-Enforced URI Processing Properties

For information about not-enforced rule evaluation and caching, see "Not-Enforced Lists".

Not-Enforced URIs

A space-delimited list of URIs that do not require authentication. The following example requires no authentication to access content in the `/public` and `/images` paths:

```
/public/* /images/*
```

Use the following format to specify not-enforced URIs in a local configuration file, where `n` is a unique, incrementing integer:

```
org.forgerock.agents.notenforced.uri.list[n]=Not enforced URI Rule
```

Percent-encode (URL-encode) spaces or other reserved characters. For example, `/my%20public%20app/`.

Use the following options to fine-tune the list of not-enforced URIs:

• Inverting Rules

- Invert specific rules in the not-enforced URI list: Precede the rule with the keyword `NOT`, separated by a space (blank) character. In the following example, any request for a `.jpg` file in the `/private` URI requires authentication:

```
NOT /private/*.jpg
```


- Invert all rules in the not-enforced URI list: Set the Invert Not-Enforced URIs property to `true`.

- **Wildcards**

- `*`: Use this wildcard to match all characters in a not-enforced URI rule, except the question mark `?` character. The wildcard 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. Therefore `*` matches `multiple/dirs`, but `mult/*/dirs` does not match `mult/dirs`.

- `-*`: Use this wildcard to match all characters in a not-enforced URI rule, except the forward slash `/` and the question mark `?` character. The wildcard cannot be escaped. Because this wildcard does not match the `/` character, it does not span multiple levels in a URI. For example:

```
/css/-*-
```

Consider the following points when using wildcards:

- Using `*` 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 match the parameters in any order that they appear in a resource URI. For example, the following not-enforced URI rule applies to any resource URI that contains a `member_level` and `location` query parameter, in any order:

```
/customers/*?*member_level=*&location=*
```

In following example, the 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 in the request, 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 about using wildcards, see [Specifying Resource Patterns with Wildcards](#).

- **Regular Expressions**

To use a regular expression in a not-enforced URI rule, add the keyword `REGEX` followed by a blank (space) character before the URI. For example:

```
REGEX https?://www\.example\.com/([^\s]+)/.*\.jpg
```

Consider the following points when using regular expressions:

- Using 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 is logged.

• HTTP Methods

To apply not-enforced rules only when the incoming request uses a specific HTTP method, add one or more of the following keywords to the not-enforced rule: `GET`, `HEAD`, `POST`, `PUT`, `PATCH`, `DELETE`, `OPTIONS`, `TRACE`.

By default, no HTTP method is specified for a rule, and all methods are not-enforced for that rule. The following example does not require authentication for any request method to `/public`:

```
/public/*
```

When one or more HTTP methods are specified for a rule, the specified methods are not-enforced, but unspecified methods are enforced and require authentication.

The following example does not require authentication for GET requests to `/public`, but does require authentication for other HTTP methods:

```
GET /public/*
```

To specify a list of methods, add a comma-delimited list of methods, followed by a blank (space) character before the URL to match. The following example does not require authentication for GET and POST requests to `/public`:

```
GET,POST /public/*  
GET,POST,PUT /examples/notenforced/*.jpg  
GET,REGEX https?://www\.example\.com/([^\s]+)/.*\.jpg
```

To invert a method, add an exclamation mark `!` character in front of it. The following example requires authentication for `POST` requests to `/public`, but does not require authentication for other HTTPS methods:

```
!POST /public/*
```

Unrecognized keywords in a rule are ignored and do not invalidate the rest of the rule.

• Cookie Values

To apply not-enforced rules when the incoming request has a named cookie with a specified value, use the following syntax:

```
COOKIE(Name/Value/Modifiers) Not Enforced URIs
```

Name

The name of the cookie to inspect for the specified value. By default, the search is case-sensitive.

Value

A string to search for in the value field of the specified cookie. By default, the search is case-sensitive.

Modifiers

One or more modifiers to change the lookup method:

c

Perform a case-insensitive search for the cookie name.

i

Perform a case-insensitive search for the cookie value.

r

Treat the string specified in *Value* as a regular expression.

The following example does not require authentication for requests to `/private/admin/images/`, when the request contains a cookie named `login_result` (case-insensitive), with the value `VALID` (case-insensitive):

```
COOKIE(login_result/VALID/ci) /private/admin/images/*
```

You can combine cookie filters with other filters, such as HTTP methods.

The following example does not require authentication for GET, POST, and PUT requests to the `/other/records/` folder, when the request contains a cookie named `internal` (case-sensitive), with a value that matches the regular expression `.*ID` (case-insensitive):

```
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; both expressions must match in order to apply. To apply the rules as a logical OR, create two separate rules.

• Header Values

To apply not-enforced rules when the incoming request has a named header with a specified value, use the following syntax:

```
HEADER(Name/Value/Modifiers) Not Enforced URIs
```

Name

The case-insensitive name of the header to inspect for the specified value.

Value

A string to search for in the value field of the specified header.

Modifiers**i**

Perform a case-insensitive search for the header value. By default, the search is case-sensitive.

r

Treat the string in **Value** as a regular expression.

The following example allows access to .txt files in `/yearly/2019/` when the request contains a header named `ID` (case-insensitive), with the value `validated` (case-insensitive):

```
HEADER(ID/validated/i) /yearly/2019/*.txt
```

You can combine cookie filters with other filters, such as HTTP methods.

The following example allows `GET`, `POST`, and `PUT` HTTP requests to HTML resources in the `/other/records/` folder when the request contains a header named `internal` (case-insensitive), with a value that matches the regular expression `.*ID` (case-insensitive):

```
GET,POST,HEADER(internal/.*ID/ri),PUT /other/records/*.html
```

Combining a `HEADER` and `COOKIE` expression in the same rule implies a logical AND; both expressions must match. 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\/([^\s]+)\.*\.(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: Empty

Type: List of URIs

Hot-swap: Yes

Property: `org.forgerock.agents.notenforced.uri.list`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.notenforced.uri`, introduced in Java Agent 5.5.2

Invert Not-Enforced URIs

When `true`, enforce policy for the URIs and patterns specified in Not-Enforced URIs instead of allowing access to them without authentication.

Caution

For security considerations, do not enable this property. Instead, ForgeRock recommends using the `NOT` keyword to invert specific rules in the Not-Enforced URI list.

Default: `false`

Type: Boolean

Hot-swap: No

Property: `org.forgerock.agents.notenforced.uri.invert.enabled`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.notenforced.uri.invert`, introduced in Java Agent 5.5.2

Not-Enforced URIs Cache Enabled

When `true`, the agent caches evaluation (hits and misses) of the Not-Enforced URIs . Enable this setting if you are configuring many rules, such as hundreds.

For more information, see "Not-Enforced Lists".

Default: `true`

Type: Boolean

Hot-swap: No

Property: `org.forgerock.agents.notenforced.uri.cache.enabled`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.notenforced.uri.cache.enable`, introduced in Java Agent 5.5.2

Not-Enforced URIs Cache Size

The maximum number of cached resource URLs that are matched by a not-enforced rule (inverted or not inverted).

Default: 1000

Type: Integer

Hot-swap: No

Property: `org.forgerock.agents.notenforced.uri.cache.size`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.notenforced.uri.cache.size`, introduced in Java Agent 5.5.2

Not Enforced Favicon

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

When `true`, the agents do not enforce access to any files named `favicon.ico`, by inserting an internal not-enforced rule of `GET */favicon.ico`.

Default: `true`

Type: Boolean

Hot-swap: No

Bootstrap property: Yes

Property: `org.forgerock.agents.auto.not.enforce.favicon.enabled`, introduced in Java Agent 5.7

Alias: `org.forgerock.agents.auto.not.enforce.favicon`, introduced in Java Agent 5.7

Refresh Session Idle Time

⚠ *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

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 information about not-enforced rule evaluation and caching, see "Not-Enforced Lists".

Not-Enforced Client List

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:

```
org.forgerock.agents.notenforced.ip.list[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

To apply not-enforced rules when the incoming request has a named cookie with a specified value, use the following syntax:

```
COOKIE(Name/Value/Modifiers) Not Enforced IPs
```

Where:

Name

The name of the cookie to inspect for the specified value.

The Agent searches for the cookie name, using a case-sensitive search. If a match is found, the cookie is returned. If the match fails, the Agent searches again, using a case-insensitive search. If a match is found the cookie is returned, and a warning is issued to the logs.

Value

A string to search 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 search for the cookie value. By default, the search is case-sensitive.

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` present on the request that has a value `VALID` case-insensitive, 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` 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: Empty

Type: List of IP addresses

Hot-swap: Yes

Property: `org.forgerock.agents.notenforced.ip.list`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.notenforced.ip`, introduced in Java Agent 5.5.2

Invert Not Enforced IPs

When `true`, enforce policy for the IPs specified in Not-Enforced Client List property instead of allowing access to them without authentication.

Caution

For security considerations, do not enable this property. Instead, ForgeRock recommends using the `NOT` keyword to invert specific rules in the Not-Enforced Client IP List.

Default: `false`

Type: Boolean

Hot-swap: No

Property: `org.forgerock.agents.notenforced.ip.invert.enabled`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.notenforced.ip.invert`, introduced in Java Agent 5.5.2

Not-Enforced IP Cache Enabled

When `true`, the agent caches evaluation (hits and misses) of the not-enforced IP list. Enable this setting if you are configuring many rules, such as hundreds.

For more information, see "Not-Enforced Lists".

Default: `true`

Type: Boolean

Hot-swap: No

Property: `org.forgerock.agents.notenforced.ip.cache.enabled`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.notenforced.ip.cache.enable`, introduced in Java Agent 5.5.2

Not-Enforced IP Cache Size

The maximum number of cached IP addresses that are matched by a not-enforced rule (inverted or not inverted).

Default: 1000

Type: Integer

Hot-swap: No

Property: `org.forgerock.agents.notenforced.ip.cache.size`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.notenforced.ip.cache.size`, introduced in Java Agent 5.5.2

Profile Attributes

Profile Attribute Fetch Mode

- **NONE**: Profile attributes are not gathered
- **HTTP_HEADER**: Profile attributes are taken from the HTTP headers
- **REQUEST_ATTRIBUTE**: Profile attributes are taken from the request attributes
- **HTTP_COOKIE**: Profile attributes are taken from HTTP cookies

Default: **NONE**

Type: Constrained string

Hot-swap: Yes

Property: `org.forgerock.agents.response.profile.fetch.mode`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.profile.attribute.fetch.mode`, introduced in Java Agent 5.5.2

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 `org.forgerock.agents.profile.attribute.map[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`.

Default: Empty

Type: Map, with format `profile attribute:HTTP header`

Hot-swap: Yes

Property: `org.forgerock.agents.profile.attribute.map`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.profile.attribute.mapping`, introduced in Java Agent 5.5.2

Response Attributes

Response Attribute Fetch Mode

- **NONE**: Response attributes are not gathered
- **HTTP_HEADER**: Response attributes are taken from the HTTP headers
- **REQUEST_ATTRIBUTE**: Response attributes are taken from the request attributes
- **HTTP_COOKIE**: Response attributes are taken from HTTP cookies

Default: **NONE**

Type: Constrained string

Hot-swap: Yes

Property: `org.forgerock.agents.response.attribute.fetch.mode`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.response.attribute.fetch.mode`, introduced in Java Agent 5.5.2

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 `org.forgerock.agents.response.attribute.map[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`.

Default: Empty

Type: Map, with format `response attribute:HTTP header`

Hot-swap: Yes

Property: `org.forgerock.agents.response.attribute.map`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.response.attribute.mapping`, introduced in Java Agent 5.5.2

Common Attributes Fetching Processing Properties

Cookie Separator Character

The separator for multiple values of the same attribute when it is set as a cookie.

Default: |

Type: String

Hot-swap: Yes

Property: `org.forgerock.agents.attribute.cookie.separator`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.attribute.cookie.separator`, introduced in Java Agent 5.5.2

Fetch Attribute Date Format

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`

Type: String (`java.text.SimpleDateFormat`)

Hot-swap: Yes

Property: `org.forgerock.agents.attribute.date.format`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.attribute.date.format`, introduced in Java Agent 5.5.2

Attribute Cookie Encode

When `true`, attribute values are URL-encoded before being set as a cookie.

Default: `true`

Type: Boolean

Hot-swap: Yes

Property: `org.forgerock.agents.attribute.cookie.encode.enabled`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.attribute.cookie.encode`, introduced in Java Agent 5.5.2

Session Attributes

Session Attribute Fetch Mode

- `NONE`: Session attributes are not gathered
- `HTTP_HEADER`: Session attributes are taken from the HTTP headers
- `REQUEST_ATTRIBUTE`: Session attributes are taken from the request attributes
- `HTTP_COOKIE`: Session attributes are taken from HTTP cookies

Default: `NONE`

Type: Constrained string

Hot-swap: Yes

Property: `org.forgerock.agents.session.attribute.fetch.mode`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.session.attribute.fetch.mode`, introduced in Java Agent 5.5.2

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 `org.forgerock.agents.session.attribute.map[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`.

Default: Empty

Type: Map, with format `session attribute:HTTP header`

Hot-swap: Yes

Property: `org.forgerock.agents.session.attribute.map`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.session.attribute.mapping`, introduced in Java Agent 5.5.2

Privilege Attributes Processing Properties

⚠ The following properties do not apply to Java Agents 5.7, although they may appear in the AM console:

Default Privileged Attribute

⚠ This property does not apply to Java Agents 5.7, although it might appear in the AM console.

Property: `com.sun.identity.agents.config.default.privileged.attribute`

Privileged Attribute Type

⚠ This property does not apply to Java Agents 5.7, although it might appear in the AM console.

Property: `com.sun.identity.agents.config.privileged.attribute.type`

Privileged Attributes To Lower Case

⚠ This property does not apply to Java Agents 5.7, although it might appear in the AM console.

Property: `com.sun.identity.agents.config.privileged.attribute.tolowercase`

Privileged Session Attribute

⚠ This property does not apply to Java Agents 5.7, although it might appear in the AM console.

Property: `com.sun.identity.agents.config.privileged.session.attribute`

Enable Privileged Attribute Mapping

⚠ This property does not apply to Java Agents 5.7, although it might appear in the AM console.

Property: `com.sun.identity.agents.config.privileged.attribute.mapping.enable`

Privileged Attribute Mapping

⚠ This property does not apply to Java Agents 5.7, although it might appear in the AM console.

Property: `com.sun.identity.agents.config.privileged.attribute.mapping`

Custom Authentication Processing Properties

⚠ The following properties do not apply to Java Agents 5.7, although they may appear in the AM console:

Custom Authentication Handler

⚠ This property does not apply to Java Agents 5.7, although it might appear in the AM console.

Property: `com.sun.identity.agents.config.auth.handler`

Custom Logout Handler

⚠ This property does not apply to Java Agents 5.7, although it might appear in the AM console.

Property: `com.sun.identity.agents.config.logout.handler`

Custom Verification Handler

⚠ This property does not apply to Java Agents 5.7, although it might appear in the AM console.

Property: `com.sun.identity.agents.config.verification.handler`

Continuous Security Properties

For 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.

The Agent searches for the cookie name, using a case-sensitive search. If a match is found, the cookie is returned. If the match fails, the Agent searches again, using a case-insensitive search. If a match is found the cookie is returned, and a warning is issued to the logs.

- `map_entry_name` specifies the name of the entry within the environmental conditions map that contains the value of `cookie_name`.

Example:

```
org.forgerock.agents.continuous.security.cookies.map[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
<code>requestIp</code> ^a	Contains the inbound request's IP address. The Java agent determines the IP as follows: <ul style="list-style-type: none"> • If the <code>com.sun.identity.agents.config.client.ip.header</code> property is configured, the agent extracts the IP address from the header. • If the <code>com.sun.identity.agents.config.client.ip.header</code> property is not configured, the agent uses the <code>HttpServletRequest.getRemoteAddr</code> Java function to determine the IP address.
<code>requestDNSName</code> ^b	Contains the inbound request's host name. The Java agent determines the host name as follows: <ul style="list-style-type: none"> • If the <code>com.sun.identity.agents.config.client.hostname.header</code> property is configured, the agent extracts the host name from the header. • If the <code>com.sun.identity.agents.config.client.hostname.header</code> property is not configured, the agent uses the <code>HttpServletRequest.getRemoteHost</code> Java function to determine the host name.
<code>variable_name</code> ^c	Contains an array of cookie or header values.

^aThe `requestIp` entry is created in the map regardless of how the continuous security properties are configured.

^bThe `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.agents.continuous.security.cookies.map[ssid]=mySSID
org.forgerock.agents.continuous.security.headers.map[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, `77xe99f4zqill99z`.
- `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: Empty

Type: Map

Hot-swap: Yes

Property: `org.forgerock.agents.continuous.security.cookies.map[cookie_name]=map_entry_name`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.openam.agents.config.continuous.security.cookies`, introduced in Java Agent 5.5.2

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.agents.continuous.security.headers.map[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
<code>requestIp</code> ^a	<p>Contains the inbound request's IP address. The Java agent determines the IP as follows:</p> <ul style="list-style-type: none"> • If the <code>com.sun.identity.agents.config.client.ip.header</code> property is configured, the agent extracts the IP address from the header. • If the <code>com.sun.identity.agents.config.client.ip.header</code> property is not configured, the agent uses the <code>HttpServletRequest.getRemoteAddr</code> Java function to determine the IP address.
<code>requestDNSName</code> ^b	<p>Contains the inbound request's host name. The Java agent determines the host name as follows:</p> <ul style="list-style-type: none"> • If the <code>com.sun.identity.agents.config.client.hostname.header</code> property is configured, the agent extracts the host name from the header. • If the <code>com.sun.identity.agents.config.client.hostname.header</code> property is not configured, the agent uses the <code>HttpServletRequest.getRemoteHost</code> Java function to determine the host name.
<code>variable_name</code> ^c	Contains an array of cookie or header values.

^aThe `requestIp` entry is created in the map regardless of how the continuous security properties are configured.

^bThe `requestDNSName` entry is created in the map regardless of how the continuous security properties are configured.

^cThere may be as many `variable_name` entries as values specified in the continuous security properties.

Consider the following example:

```
org.forgerock.agents.continuous.security.cookies.map[ssid]=mySsid
org.forgerock.agents.continuous.security.headers.map[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, `77xe99f4zqill99z`.
- `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: Empty

Type: Map

Hot-swap: Yes

Property: `org.forgerock.agents.continuous.security.headers.map[header_name]=map_entry_name`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.openam.agents.config.continuous.security.headers`, introduced in Java Agent 5.5.2

Query Parameter Handling Properties

For information about how the Java agent handles query parameters, see "Query Parameter Handling".

Remove Query Parameters

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

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.

- `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.agents.unwanted.http.url.param.list[0]=myapp.example.com/customers|location,lang
org.forgerock.agents.unwanted.http.url.param.list[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.

Default: Empty

Type: List of HTTP query parameter names

Hot-swap: Yes

Property: `org.forgerock.agents.unwanted.http.url.param.list[n]`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.openam.agents.config.conditional.unwanted.http.url.params`, introduced in Java Agent 5.5.2

Regular Expression Remove Query Parameters

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

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.

- **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.agents.unwanted.http.url.params.list.regex[0]=myapp.example.com|b.*,gp(a|p|s),  
org.forgerock.agents.unwanted.http.url.params.list.regex[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.

Default: Empty

Type: List of HTTP query parameter names

Hot-swap: Yes

Property: `org.forgerock.agents.unwanted.http.url.params.regex.list[n]`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.agents.unwanted.http.url.params.regex.list`, introduced in Java Agent 5.5.2, and `org.forgerock.openam.agents.config.conditional.unwanted.http.url.params.regex`, introduced in Java Agent 5.5.2

Retain Query Parameters

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

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.
- `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.agents.wanted.http.url.param.list[0]=myapp.example.com/news|area
org.forgerock.agents.wanted.http.url.param.list[1]=example.com/news|area,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.

Default: Empty

Type: List of HTTP query parameter names

Hot-swap: Yes

Property: `org.forgerock.agents.wanted.http.url.param.list[n]`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.openam.agents.config.conditional.wanted.http.url.params`, introduced in Java Agent 5.5.2

Regular Expression Retain Query Parameters

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

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.

- `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.agents.wanted.http.url.params.list.regex[0]=example.com/market|regist.*
org.forgerock.agents.wanted.http.url.params.list.regex[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.

Default: Empty

Type: List of HTTP query parameter names

Hot-swap: Yes

Property: `org.forgerock.agents.wanted.http.url.params.list.regex[n]`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.agents.wanted.http.url.params.regexp.list`, introduced in Java Agent 5.5.2, and `org.forgerock.openam.agents.config.conditional.wanted.http.url.params.regexp`, introduced in Java Agent 5.5.2

Authentication Failure Properties

Authentication Fail Reason Url

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The URL or URI to which the agent redirects the user after a failed authentication. For example, `app/AuthFailed.html`.

Note that authentication failure can happen for reasons other than wrong credentials. For more information, see "Authentication Failure Notification".

If this property is not set, the agent redirects the user to the URL defined in the Goto URL property. If both are unset, the agent returns an HTTP 400 message.

To configure the agent to send the reason for authentication failure in a query parameter, configure Login Reason Parameter Name.

Default: Empty

Type: URI/URL string

Property: `org.forgerock.agents.authn.fail.url`, introduced in Java Agent 5.7

Authentication Fail Reason Parameter Name

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

Specifies a query parameter name to contains the reason why authentication failed. The agent appends this parameter to the URL or URI defined by Authentication Fail Reason Url .

For example, if the value of property is `reason`, the agent redirects the end user to a URL similar to the following:

```
http://myapp.example.com:8080/app/AuthFailed.html?reason=AUTHN_BOOKKEEPING_COOKIE_MISSING
```

If not set, the agent does not append the reason for the authentication failure, when redirecting to the URL or URI. This property requires Authentication Fail Reason Url to be set.

Caution

For increased security, remap the reason for authentication failure, using Authentication Fail Reason Parameter Value Map.

Default: Empty

Type: String

Property: `org.forgerock.agents.authn.fail.reason.parameter.name`, introduced in Java Agent 5.7

Authentication Fail Reason Parameter Value Map

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

Caution

After an authentication failure, malicious users can use the information you expose to gain access to the system. Map the reason for authentication failure to something generic, or something that is meaningful inside your organization.

Map reasons for authentication failure to custom messages, as follows:

Authentication Failure Reasons

Reason Code	Meaning
<code>AUTHN_BOOKKEEPING_COOKIE_MISSING</code>	The agent cannot find the authentication tracking cookie (defined in the <code>org.forgerock.agents.authn.cookie.name</code> property). This error can happen if the user successfully authenticates, but clicks the back button of the browser to return to the previous page.
<code>NONCE_MISSING</code>	The agent found the authentication tracking cookie, but it cannot find the unique identifier of the authentication request inside the cookie. This error can happen if the user successfully authenticates, but clicks the back button of the browser to return to the previous page.
<code>BAD_AUDIENCE</code>	The audience in the JWT did not correspond to the audience in the cookie entry. This error can happen if all agents working in a cluster do not have the same Agent Profile Name.
<code>NO_TOKEN</code>	The agent cannot find the session ID token.
<code>TOKEN_EXPIRED</code>	The agent found the session ID token, but it is past its expiry date.

Reason Code	Meaning
AM_SAYS_INVALID	The agent found the session ID token, the expiry time is correct, but AM returns that the ID token is invalid.
JWT_INVALID	The agent found the session ID token, but cannot parse it.
EXCEPTION	This reason can have the following meanings: <ul style="list-style-type: none"> The agent found the session ID token, but threw an exception while parsing it. The agent cannot connect to AM to validate the ID token, maybe due to a network outage.

Specify the authentication failure reason from the preceding table as the map key, and your custom error identifier string as the value. For example:

```
org.forgerock.agents.authn.fail.reason.remapper[TOKEN_EXPIRED]=MY_ERROR_MESSAGE
```

Consider remapping all the failure reasons to a new error message, for example, `ERROR`, then be specific on those that hold more meaning for your environment. For example:

```
org.forgerock.agents.authn.fail.reason.remapper=ERROR
org.forgerock.agents.authn.fail.reason.remapper[AUTHN_BOOKKEEPING_COOKIE_MISSING]=BACK_BUTTON_PRESSED
org.forgerock.agents.authn.fail.reason.remapper[NONCE_MISSING]=BACK_BUTTON_PRESSED
```

To map all the authentication failure reasons to the same message, you do not need to specify a key in the property.

This property requires `Authentication Fail Reason Url` to be set.

Default: Empty

Type: Map, keyed by reason for failure

Property: `org.forgerock.agents.authn.fail.reason.remapper[REASON]`, introduced in Java Agent 5.7

Goto Url

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The URL to which the agent redirects the user when required redirection properties are not set in the agent configuration.

For example, after an authentication failure, if the property `Authentication Fail Reason Url` is not set, the agent redirects the user to the URL defined in this property. If both are unset, the agent returns an HTTP 400.

Default: Empty

Type: String, or an URL or URI of an HTML or other page

Hot-swap: Yes

Property: `org.forgerock.agents.default.goto.url`, introduced in Java Agent 5.7

Alias: `com.sun.identity.agents.config.openam.agent.default_goto_url`, introduced before Java Agent 5.5.2

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
- Caching Properties
- Cross-Domain SSO Properties
- Cookie Reset Properties

Cookie Properties

Cookie Name

🔒 *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

Property: `com.iplanet.am.cookie.name`

Http Only

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

When `true`, cookies are flagged as HTTPOnly. Use this property prevent scripts and third-party programs from accessing the cookies.

Default: `true`

Type: Boolean

Hot-swap: Yes

Property: `com.sun.identity.cookie.httponly`, introduced in Java Agent 5.5.2

Encode Cookies

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

When `true`, cookies are encoded.

Default: `false`

Type: Boolean

Hot-swap: Yes

Property: `com.iplanet.am.cookie.encode`, introduced in Java Agent 5.5.2

Pre-Authenticated Cookie Name

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The name of the bookkeeping "pre-authentication" cookie the agent uses for unauthenticated requests, to check that the authentication URL (used to invoke AM for authentication) isn't being used in a replay attack.

- (Default) One cookie is used for all unauthenticated requests.

Dozens or even hundreds of requests can be bundled into the same cookie, and if the number of concurrent requests is large enough, the cookie can exceed the browser size limit and requests can fail.

As a workaround, add as many resources as possible to the not-enforced lists, because these are not tracked in the cookie.

- (From version 5.7) One cookie is used for each unauthenticated request The cookie name contains the secure ID in a user-defined manner.

Place the `%n` string once in the name of the cookie. For example:

```
org.forgerock.agents.authn.cookie.name=pre-authn-%n
org.forgerock.agents.authn.cookie.name=pre-%n-authn
org.forgerock.agents.authn.cookie.name=%n-pre-authn
org.forgerock.agents.authn.cookie.name=%n
```


Alternatively, set the cookie name to `%n`. The agent translates the string to a unique identifier when it creates the cookie.

Caution

Use this mode only if required; creating additional cookies can impact performance.

The following information is stored in the cookie:

- URL of the original request
- HTTP mode
- Secure ID (subsequently baked into the nonce of the returned JWT)
- Audience
- Relevant ACR information
- Transaction ID
- Authentication bookkeeping cookie expiry time

The cookie is compressed and finally signed.

Default: `amFilterCDSSORequest`

Type: String

Hot-swap: Yes

Property: `org.forgerock.agents.authn.cookie.name`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.config.cdsso.cookie.name`, introduced in Java Agent 5.5.2

Pre-Authenticated Cookie Max Age

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The maximum age in seconds of the authentication bookkeeping cookie.

Default: `300`

Type: Integer

Hot-swap: Yes

Property: `org.forgerock.agents.authn.cookie.max.age.seconds`, introduced in Java Agent 5.6.3

Caching Properties

~~SSO Cache Enable~~

⚠ *This property does not apply to Java Agents 5.7, although it might 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 5.7, although it might 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.

Caution

Changing the value of this property while the Agent is running prevents it from functioning if OIDC authentication is being used. The Agent must be restarted as soon as the value in AM has been altered and the properties saved.

Default: None. The value of this property must be provided by AM.

Type: URI

Hot-swap: No

Property: `org.forgerock.agents.authn.redirect.uri`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.cdsso.redirect.uri`, introduced in Java Agent 5.5.2

~~CDSSO Servlet URL~~

⚠ *This property does not apply to Java Agents 5.7, although it might 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 5.7, although it might 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 5.7, although it might appear in the AM console.*

Property: `com.sun.identity.agents.config.cdsso.trusted.id.provider`

Secure Cookies Enable

When `true`, all cookies written by the agent are secure. For backward compatibility, the default is `false`.

Default: `false`

Type: Boolean

Hot-swap: Yes

Property: `org.forgerock.agents.secure.cookies.enabled`, introduced in Java Agent 5.5.2

Alias: `com.iplanet.am.cookie.secure`, introduced in Java Agent 5.5.2, and `com.sun.identity.agents.config.cdsso.secure.enable`, introduced in Java Agent 5.5.2

Cookie Reset Properties

Cookie Reset

When `true`, Java agents reset cookies in the response before redirecting to authentication.

This property is linked to the properties `Profile Attribute Fetch Mode` and `Session Attribute Fetch Mode`. When either of these is set to `HTTP_COOKIE`, the Agent builds a list of cookies.

Default: `false`

Type: Boolean

Hot-swap: Yes

Property: `org.forgerock.agents.cookie.reset.enabled`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.cookie.reset.enable`, introduced in Java Agent 5.5.2

Cookie Reset Name List

List of cookies to reset if Cookie Reset is `true`.

The Agent searches for the cookie name, using a case-sensitive search. If a match is found, the cookie is returned. If the match fails, the Agent searches again, using a case-insensitive search. If a match is found the cookie is returned, and a warning is issued to the logs.

Default: Empty

Type: List of cookie names

Hot-swap: Yes

Property: `org.forgerock.agents.cookie.reset.name.list`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.cookie.reset.name`, introduced in Java Agent 5.5.2

Cookie Reset Domain Map

Specifies how names from Cookie Reset Name List correspond to cookie domain values when the cookie is reset.

Default: Empty

Type: Map, with format *cookie name:cookie domain*

Hot-swap: Yes

Property: `org.forgerock.agents.cookie.reset.domain.map`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.cookie.reset.domain`, introduced in Java Agent 5.5.2

Cookie Reset Path Map

Specifies how names from Cookie Reset Name List correspond to cookie paths when the cookie is reset.

Default: Empty

Type: Map, with format *cookie name:cookie path*

Hot-swap: Yes

Property: `org.forgerock.agents.cookie.reset.path.map`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.cookie.reset.path`, introduced in Java Agent 5.5.2

Set-Cookie Internal Map

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

When creating internal cookies, such as `am-auth-jwt` and the bookkeeping "pre-authentication" cookies, set additional attributes by adding text into the Set-Cookie header.

Specify a `key:value` map, where the key is the cookie name, and the value the string to add to the Set-Cookie header. If the key is omitted, the value becomes the default for all cookies.

If the string is not suitable for the Set-Cookie header, the Agent might fail to create the cookie. Separate multiple values with a semicolon.

The following property setting sets the `SameSite` attribute of the `am-auth-jwt` cookie:

```
org.forgerock.agents.set.cookie.internal.map[am-auth-jwt]=samesite=strict
```

The following example sets the `SameSite` attribute of all cookies:

```
org.forgerock.agents.set.cookie.internal.map=samesite=strict
```

The following example sets several attributes of `myCookie`:

```
org.forgerock.agents.set.cookie.internal.map[myCookie]=Max-Age=10000; Domain=.my.default.fqdn
```

Default: Empty

Type: Map, with format *cookie name: string ; ...*

Hot-swap: Yes

Property: `org.forgerock.agents.set.cookie.internal.map`, introduced in Java Agent 5.6.3

Set-Cookie Attribute Map

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

When creating cookies with the `AttributeTaskHandler`, out of `Profile Attributes`, `Response Attributes`, and `Session Attributes`, set additional attributes by adding text into the Set-Cookie header.

Specify a `key:value` map, where the key is the cookie name, and the value the string to add to the the Set-Cookie header.

If the string is not suitable for the Set-Cookie header, the Agent might fail to create the cookie.

Separate multiple values with a semicolon.

The following example sets the `SameSite` attribute of the `myCookie` cookie:

```
org.forgerock.agents.set.cookie.attribute.map[myCookie]=samesite=strict
```

The following example sets several attributes of `myCookie`:

```
org.forgerock.agents.set.cookie.attribute.map[myCookie]=Max-Age=10000; Domain=.my.default.fqdn
```

Default: Empty

Type: Map, with format *cookie name: string ; ...*

Hot-swap: Yes

Property: `org.forgerock.agents.set.cookie.attribute.map`, introduced in Java Agent 5.6.3

Samesite Cookie Attributes Excluded User Agents Pattern List

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

List of user agents excluded from receiving SameSite cookie attributes.

The following example gives the default list of user agents excluded from receiving SameSite cookie attributes:

```
org.forgerock.agents.samesite.excluded.user.agents.list[0]=(?!Chrom(e|ium))Version/(1[0-2]|[1-9])(\.\d+)*.*Safari/
org.forgerock.agents.samesite.excluded.user.agents.list[1]=(iP.+; CPU .05 (1[0-2]|[1-9])_[\d].*)
AppleWebKit/
org.forgerock.agents.samesite.excluded.user.agents.list[2]=^Mozilla/[\.\d]+ (Macintosh;.Mac OS X
10_14[\d]) AppleWebKit/[\.\d](KHTML, like Gecko)$
org.forgerock.agents.samesite.excluded.user.agents.list[3]=Chrom(e|ium)[^ /]/(5[1-9]|6[0-6])\.[\.\d]
org.forgerock.agents.samesite.excluded.user.agents.list[4]=UCBrowser/(12\.13\.[01][^\d]|12\.1[0-2]\.
12\.[0-9]\.[1[01]\.|[0-9]\.)[\.\d]*
```

To specify different user agent patterns, add them in AM as custom properties, as described in [Custom Properties](#). When user agent patterns are specified, the default list of user agents is ignored.

When the following example is specified in AM as a custom property, Chrome versions 67-89 are excluded from receiving SameSite cookie attributes. However, the Chrome versions in the default list of user agents are no longer excluded:

```
org.forgerock.agents.samesite.excluded.user.agents.list[0]=Chrom(e|ium)[^ /]/(6[7-9]|8[0-9])\.[\.\d]
```

Type: List of expressions representing user agents

Hot-swap: Yes

Property: `org.forgerock.agents.samesite.excluded.user.agents.list`, introduced in Java Agent 5.6.3

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 information, see "Redirection and Conditional Redirection".

Allow Custom Login Mode

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

Sets the login redirection mode, as follows:

- **true**: Allow the custom login redirection mode (Non-OIDC compliant login flow). Use with the following properties:
 - OpenAM Login URL (`com.sun.identity.agents.config.login.url`)
 - `org.forgerock.agents.legacy.login.url.list`, see Custom Conditional Login URL.
- **false**: Use the default login redirection mode (OIDC compliant login flow). Use with the following properties:
 - OAuth 2.0 Login List

Before configuring this property, read "Redirection and Conditional Redirection".

Default: `false`

Type: Boolean

Hot-swap: Yes

Property: `org.forgerock.agents.legacy.login.enabled`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.openam.agents.config.allow.custom.login`, introduced in Java Agent 5.5.2

Custom Conditional Login URL

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

Allows additional parameters to be conditionally added to legacy login URLs.

Default: Empty

Type: List of strings, with format *domain/path|url?param1=value1¶m2=value2*

Hot-swap: Yes

Property: `org.forgerock.agents.legacy.login.url.list`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.openam.agents.config.conditional.custom.login.url`, introduced in Java Agent 5.5.2

Conditional Legacy Logout URL List

Allows additional parameters to be conditionally added to legacy logout URLs.

Default: Empty

Type: List of strings, with format *domain/path|url?param1=value1¶m2=value2*

Hot-swap: Yes

Property: `org.forgerock.agents.conditional.logout.url.list`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.openam.agents.config.conditional.logout.url`, introduced in Java Agent 5.5.2

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¶meter1=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 Goto Parameter Name 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 choose 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: Empty

Type: List, of which only the first element is ever used

Hot-swap: Yes

Property: `com.sun.identity.agents.config.login.url`, introduced in Java Agent 5.5.2

OpenAM Conditional Login URL

Conditionally redirect users based on the incoming request URL. If the incoming request URL matches a 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.

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 `[Domain/path][URL][?realm=value&module=value2&service=value3]`, with no spaces between values. Specify values in conditional redirects 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.
- **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¶m1=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:

```
org.forgerock.agents.oauth.login.url.list[0]=example.com|https://openam.example.com/openam/oauth2/authorize
org.forgerock.agents.oauth.login.url.list[1]=myapp.domain.com|https://openam2.example.com/openam/oauth2/authorize?realm=sales
org.forgerock.agents.oauth.login.url.list[2]=sales.example.com/marketplace|?realm=marketplace
org.forgerock.agents.oauth.login.url.list[3]=|https://openam3.example.com/openam/oauth2/authorize?realm=customers&module=myAuthModule
```

Default: Empty

Type: List of strings, with format `domain/path|url?param1=value1¶m2=value2`

Hot-swap: Yes

Property: `org.forgerock.agents.oauth.login.url.list[n]=[Domain]||[URL][?realm=value&module=value2&service=value3]`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.openam.agents.config.conditional.login.url`, introduced in Java Agent 5.5.2

OAuth 2.0 Login List

Conditionally redirect users based on the incoming request URL. If the incoming request URL matches a 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.

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 OpenAM 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 `[Domain/path] | [URL?realm=value¶meter1=value1...]`, with no spaces between values. Specify values in conditional redirects 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.
- `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.

- `¶meter1=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¶meter1=value1¶meter2=value2`.

Important

When the agent redirects the user to the custom login page, it appends a `goto` parameter (as configured in the Goto Parameter Name 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:

```
org.forgerock.agents.legacy.login.url.list[0]=example.com|https://openam.example.com/openam/XUI/
#login&realm=customers
org.forgerock.agents.legacy.login.url.list[1]=myapp.domain.com|https://login.example.com/apps/
login.jsp?realm=sales
org.forgerock.agents.legacy.login.url.list[2]=sales.example.com/marketplace|?realm=marketplace
org.forgerock.agents.legacy.login.url.list[3]=|https://login.example.com/apps/login.jsp?
realm=sales&isblue=true&carowner=true
```

Default: Empty

Type: List of strings, with format `domain/path|url?param1=value1¶m2=value2`

Hot-swap: Yes

Property: `org.forgerock.agents.oauth.login.url.list[n]=[Domain]|[URL][?realm=value&module=value2&service=value3]`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.openam.agents.config.conditional.login.url`, introduced in Java Agent 5.5.2

Login Reason Parameter Name

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

When the agent is configured in "Custom Redirection Login Mode", this property specifies the name of a parameter included in calls to the custom login URL, to indicate why authentication is required. The parameter value can be used in a custom login page to provide additional feedback to the authenticating user.

If this property is specified, the agent includes a parameter named with the property value, and including one of the following values:

- **NO_TOKEN**: No token present in the original request.
- **TOKEN_EXPIRED**: Expiry time of the JWT was in the past.
- **EXCEPTION**: An unknown exception occurred either while parsing the JWT or at some other stage of authentication.

Tip

To limit the risk of leaking useful information about the authorization process, use Login Reason Value Map to change the strings for the above values.

For example, specifying `org.forgerock.agents.login.reason.parameter.name=auth_reason` might cause the agent to redirect authentication to the following URL:

```
https://custom.example.com:8443/.../login_endpoint?...&auth_reason=TOKEN_EXPIRED&...
```

Do not enter a value that clashes with other parameters used for authentication; for example, `realm` or `goto`.

Default: Empty

Type: String

Hot-swap: Yes

Property: `org.forgerock.agents.login.reason.parameter.name`, introduced in Java Agent 5.7

Login Reason Value Map

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

When Login Reason Parameter Name is configured, this property specifies alternative strings to use for the supported values.

For example:

```
org.forgerock.agents.login.reason.map[NO_TOKEN]=notoken  
org.forgerock.agents.login.reason.map[TOKEN_EXPIRED]=expired  
org.forgerock.agents.login.reason.map[EXCEPTION]=exception
```

With the above configuration, using `org.forgerock.agents.login.reason.parameter.name=auth_reason` might cause the agent redirect authentication to the following URL:

```
https://custom.example.com:8443/.../login_endpoint?...&auth_reason=notoken&...
```

Default: Empty

Type: Map, keyed by reason for failure

Property: `org.forgerock.agents.login.reason.map`, introduced in Java Agent 5.7

Alias: `org.forgerock.agents.org.forgerock.agents.login.reason.remapper`, introduced in Java Agent 5.7

~~Login URL Prioritized~~

☞ *This property does not apply to Java Agents 5.7, although it might 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 5.7, although it might 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 5.7, although it might 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 5.7, although it might 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 URL. If the incoming request URL matches a 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 `[Domain/path][URL][?realm=value]`, with no spaces between values. Specify values in conditional redirects 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.

- `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/realm/root/realm/customers/authorize` and do not add the `realm` parameter.

Examples:

```
org.forgerock.agents.conditional.logout.url.list[0]=example.com|https://openam.example.com/openam/
oauth2/authorize
org.forgerock.agents.conditional.logout.url.list[2]=sales.example.com/marketplace|?realm=marketplace
org.forgerock.agents.conditional.logout.url.list[3]=|https://openam3.example.com/openam/oauth2/
authorize?realm=customers
```

Default: `false`

Type: List of strings, with format `domain/path|url?param1=value1¶m2=value2`

Hot-swap: Yes

Property: `org.forgerock.agents.conditional.logout.url.list[n]`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.openam.agents.config.conditional.logout.url`, introduced in Java Agent 5.5.2

Logout URL Prioritized

⚠ This property does not apply to Java Agents 5.7, although it might 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 5.7, although it might 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 5.7, although it might appear in the AM console.

Property: `com.sun.identity.agents.config.logout.url.probe.timeout`

Authentication Service Properties

AM Protocol

Specifies the protocol used by the AM server.

Note that this is a bootstrap property. If you need to change it manually, configure it in the `OpenSSOAgentBootstrap.properties` file.

Default: Set by installer

Type: Constrained string, probably `http` or `https`.

Bootstrap property: Yes

Hot-swap: No

Property: `org.forgerock.agents.am.protocol`, introduced in Java Agent 5.5.2

Alias: `com.ipplanet.am.server.protocol`, introduced in Java Agent 5.5.2

AM Host

Specifies the AM server host name.

Note that this is a bootstrap property. If you need to change it manually, configure it in the `OpenSSOAgentBootstrap.properties` file.

Default: Set by installer

Type: String

Bootstrap property: Yes

Hot-swap: no

Property: `org.forgerock.agents.am.hostname`, introduced in Java Agent 5.5.2

Alias: `com.ipplanet.am.server.host`, introduced in Java Agent 5.5.2

AM Port

Specifies the AM AM server port number.

Note that this is a bootstrap property. If you need to change it manually, configure it in the `OpenSSOAgentBootstrap.properties` file.

Default: Set by installer

Type: Integer

Bootstrap property: Yes

Hot-swap: no

Property: `org.forgerock.agents.am.port`, introduced in Java Agent 5.5.2

Alias: `com.ipplanet.am.server.port`, introduced in Java Agent 5.5.2

AM Path

Specifies the path to the AM server.

Note that this is a bootstrap property. To change it manually, configure it in the `OpenSSOAgentBootstrap.properties` file.

Default: Set by installer

Type: String

Bootstrap property: Yes

Hot-swap: No

Property: `org.forgerock.agents.am.path`, introduced in Java Agent 5.5.2

Alias: `com.iplanet.am.services.deploymentDescriptor`, introduced in Java Agent 5.5.2

AM Public URL

The AM Protocol, AM Host, AM Port, and AM Path, assemble into a single "private" URL of AM or a load balancer fronting a cluster of AM instances. The "private" URL is used by the Agent for tasks such as establishing websockets, and obtaining authentication tokens or session information. The AM or load balancer instance can be behind a firewall to which the Agent has access.

This property is the assembled "public" URL of AM. The "public" URL is used by the Agent to redirect the user's browser to AM for login (customised or not), or exchange an SSO token for a JWT.

Define this property when public access to AM is restricted to a different URL from the "private" URL.

Note that this is a bootstrap property. To change it manually, configure it in the `OpenSSOAgentBootstrap.properties` file.

Default: Empty, takes the value of the "private" URL

Type: URL

Bootstrap property: Yes

Hot-swap: No

Property: `org.forgerock.agents.public.am.url`, introduced in Java Agent 5.6.3

Alias: `com.forgerock.agents.public.am.url`, introduced in Java Agent 5.6.3

AM Encryption Digest

The hashing algorithm used internally by the Agent. The algorithm is present only in the configuration files, and cannot be set from the XUI.

Note that this is a bootstrap property. If you need to change it manually, configure it in the `OpenSSOAgentBootstrap.properties` file.

Changing this property it is not recommended, because it can cause the Agent to fail randomly.

Default: SHA256

Type: String constrained by hashing algorithm type, for example, MD5 (not recommended), SHA1 (not recommended), SHA256, SHA512, PBKDF2.

Bootstrap property: Yes

Hot-swap: No

Property: `org.forgerock.openam.encryption.key.digest`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.openam.encryption.key.digest`, introduced in Java Agent 5.5.2

AM Encryption Key

The key used to encrypt the Agent password. The key is set during installation process.

The property is present only in the configuration files, and cannot be set from the XUI.

To change the key after installation, manually invoke the Agent installer, with the correct options, and then manually edit the result into the configuration files.

Default: Set by the installer in the bootstrap properties file

Type: Cryptographically secure random string

Bootstrap property: Yes

Hot-swap: No

Property: `am.encryption.pwd`, introduced in Java Agent 5.6.2.1

AM Encryption Class

⚠ *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

The Java class used to encrypt the Agent password. During installation, the class is set in the bootstrap properties file as `AESWrapEncryption`, a particularly secure encryption mechanism.

The property is present only in the configuration files, and cannot be set from the XUI.

To change the class after installation, ensure that the class is available at runtime, regenerate the Agent password using the Agent installer and manually edit the newly generated encrypted password into the bootstrap property file.

Default: Set during installation in the bootstrap properties file. If manually removed or not set in bootstrap properties, it defaults to `com.ipplanet.services.util.JCEEncryption`, which is a less secure encryption mechanism.

Type: Cryptographically secure random string

Bootstrap property: Yes

Hot-swap: No

Property: `com.ipplanet.security.encryptor`, introduced in Java Agent 5.5.2

Policy Client Service Properties

Policy Evaluation 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)

Type: Map of *application name:realm*. Before version 5.6.2.1 this property was a string.

Hot-swap: Yes

Property: `org.forgerock.agents.policy.evaluation.realm.map`, introduced in Java Agent 5.6.2.1

Alias: `org.forgerock.openam.agents.config.policy.evaluation.realm`, introduced in Java Agent 5.5.2

Policy Set (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`

Type: Map of *application name:policy set*.

Hot-swap: Yes

Property: `org.forgerock.agents.policy.set.map`, introduced in Java Agent 5.6.2.1

Alias: `org.forgerock.openam.agents.config.policy.evaluation.application`, introduced in Java Agent 5.5.2

Policy Cache TTL

The time in minutes after which entries in the policy cache timeout and are purged.

Default: 3

Type: Integer

Hot-swap: No

Property: `org.forgerock.agents.policy.cache.ttl.minutes`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.polling.interval`, introduced in Java Agent 5.5.2

Session Cache TTL

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The time in minutes after entries in the session cache timeout and are purged.

If an entry is not cached, the agent must retrieve session information from AM. Therefore, by default the timeout is much longer than for the policy cache.

If not set, the value is five times the value of Policy Cache TTL.

Default: 15

Type: Integer

Hot-Swap: No

Property: `org.forgerock.agents.session.cache.ttl.minutes`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.openam.agents.config.active.session.cache.ttl.minutes`, introduced in Java Agent 5.5.2

Policy Cache Size

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The maximum number of sessions (distinct users) that can stored in the policy evaluation cache at one time.

Default: 5000

Type: Integer

Hot-Swap: No

Property: `org.forgerock.agents.policy.cache.session.size`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.policy.client.cachedSessionCap`, introduced in Java Agent 5.5.2

Policy Cache Per User

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The maximum number of distinct policy evaluation entries allowed for each session in the policy evaluation cache. The number of policy evaluation results that can be stored is Policy Cache Size multiplied by this property.

The maximum number of policy evaluation results that can be stored is the value of this property multiplied by the value of Policy Cache Size.

Default: 50

Type: Integer

Hot-Swap: No

Property: `org.forgerock.agents.policy.cache.per.session.size`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.policy.client.cacheResultsPerUserCap`, introduced in Java Agent 5.5.2

Policy Client Cache Mode

ⓘ This property does not apply to Java Agents 5.7, although it might 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 5.7, although it might appear in the AM console.

Property: `com.sun.identity.policy.client.booleanActionValues`

Policy Client Resource Comparators

ⓘ This property does not apply to Java Agents 5.7, although it might appear in the AM console.

Property: `org.forgerock.agents.policy.client.resourceComparators`, first deprecated in Java Agent 5.5.2

Alias: `com.sun.identity.policy.client.resourceComparators`, first deprecated in Java Agent 5.5.2

Restrict To Realm

+ Not available in the console for AM 6.5.x and earlier versions.

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

A map of application name by realm, to allow only users from the specified realms to access the specified application.

Default: Empty; no restriction by realm

Type: Map, with format *application name:realm*

Hot-swap: Yes

Property: `org.forgerock.agents.restrict.to.realm.map`, introduced in Java Agent 5.6.2.1

Policy Client Clock Skew

☞ *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

Property: `com.sun.identity.policy.client.clockSkew`

Continuous Security GET List

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: Empty

Type: List

Hot-swap: Yes

Property: `org.forgerock.agents.continuous.security.get.list`, introduced in Java Agent 5.7

Alias: `com.sun.identity.agents.config.policy.env.get.param`, introduced in Java Agent 5.5.2

Continuous Security POST List

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: Empty

Type: List

Hot-swap: Yes

Property: `org.forgerock.agents.continuous.security.post.list`, introduced in Java Agent 5.7

Alias: `com.sun.identity.agents.config.policy.env.post.param`, introduced in Java Agent 5.5.2

Continuous Security HTTP Session List

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: Empty

Type: List

Hot-swap: Yes

Property: `org.forgerock.agents.continuous.security.http.session.list`, introduced in Java Agent 5.7

Alias: `com.sun.identity.agents.config.policy.env.jsession.param`, introduced in Java Agent 5.5.2

Use HTTP-Redirect for composite advice

🔗 This property does not apply to Java Agents 5.7, although it might appear in the AM console.

Property: `org.forgerock.agents.policy.advice.use.redirect.enabled`, first deprecated in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.policy.advice.use.redirect`, first deprecated in Java Agent 5.5.2

Composite Advice Encode

+ Not available in the console for AM 6.5.x and earlier versions.

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

Whether to base64 URL-encode composite advices before sending them to custom login endpoints. Set to `true` to increase the security, and protect against cross-site scripting attacks.

The following example is a request from the Agent to AM for a custom login page:

- When this property is `false`, advices are not encoded:

```
http://customloginpage.agent.example.com:9191/?composite_advice=
&Advices>
  &AttributeValuePair>
    &Attribute name="AuthSchemeConditionAdvice"/>
    &Value>/:LDAP&/Value>
  &/AttributeValuePair>
&/Advices>
&goto=https://agent.example.com:443/app/post-authn-redirect?nonce=ZYZ...MJY
&resourceURL=https://agent.example.com:443/app/page
```

- When this property is `true`, advices are encoded as follows:

```
http://customloginpage.agent.example.com:9191/?composite_advice=PEFkdmd...ZXM-
&goto=https://agent.example.com:443/app/post-authn-redirect?nonce=84Y...E6w
&resourceURL=https://agent.example.com:443/app/page
```

Default: `false`

Type: Boolean

Hot-swap: Yes

Property: `org.forgerock.agents.advice.b64.url.encode`, introduced in Java Agent 5.6.2.1

Alias: `com.forgerock.agents.advice.b64.url.encode`, introduced in Java Agent 5.5.2

User Data Cache Service Properties

Enable Notification of User Data Caches

⚠ This property does not apply to Java Agents 5.7, although it might 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 5.7, although it might 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 5.7, although it might appear in the AM console.

Property: `com.sun.identity.sm.notification.enabled`

Service Data Cache Time

⚠ This property does not apply to Java Agents 5.7, although it might appear in the AM console.

Property: `com.sun.identity.sm.cacheTime`

Session Client Service Properties

Client Polling Period

⚠ This property does not apply to Java Agents 5.7, although it might 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.

- "Locale Properties"
- "Port Check Processing Properties"
- "Bypass Principal List Properties"

- "Deprecated Agent Properties"

Locale Properties

Locale Language

Default language for the agent.

Default: `en`

Type: String (locale language)

Hot-swap: No

Property: `org.forgerock.agents.locale.language`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.locale.language`, introduced in Java Agent 5.5.2

Locale Country

The default country for the agent.

Default: `US`

Type: String (locale country)

Hot-swap: no

Property: `org.forgerock.agents.locale.country`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.locale.country`, introduced in Java Agent 5.5.2

Port Check Processing Properties

Port Check Enable

When `true`, activate port checking, correcting requests on the wrong port.

Default: `false`

Type: Boolean

Hot-swap: Yes

Property: `org.forgerock.agents.port.check.enabled`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.port.check.enable`, introduced in Java Agent 5.5.2

Port Check File

🔗 *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

The name of the file containing the content to handle requests on the wrong port when port checking is enabled.

Default: `true.txt`

Type: String

Hot-swap: Yes

Property: `org.forgerock.agents.port.check.file`, first deprecated in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.port.check.file`, first deprecated in Java Agent 5.5.2

Port Check Setting

A map of port to protocol (HTTP/HTTPS). The Java agent uses the map when handling requests with invalid port numbers during port checking.

Default: `8080 HTTP`

Type: Map, with format `port number:protocol`

Hot-swap: Yes

Property: `org.forgerock.agents.port.check.map`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.port.check.setting`, introduced in Java Agent 5.5.2

Bypass Principal List Properties

~~Bypass Principal List~~

🚫 *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

Property: `com.sun.identity.agents.config.bypass.principal`

Deprecated Agent Properties

Goto Parameter Name

Renames the `goto` parameter. During redirection, the Java agent appends the requested URL to the renamed parameter. Use this property when your application requires a parameter other than `goto`, however, its use is not recommended.

In the following example, the parameter is renamed to `goto2`:

```
com.sun.identity.agents.config.redirect.param=goto2
```

The redirection URL becomes something like this:

```
https://www.example.com:8443/accessDenied.html?goto2=http%3A%2F%www.example.com%3A8020%managers%2Findex.jsp
```

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, configured with the `org.forgerock.agents.access.denied.uri.map` property.

Default: `goto`

Type: String

Hot-swap: Yes

Property: `com.sun.identity.agents.config.redirect.param`, introduced in Java Agent 5.5.2

Legacy User Agent Support Enable

☞ *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

When `true`, provide support for legacy browsers.

Default: `false`

Type: Boolean

Hot-swap: Yes

Property: `org.forgerock.agents.legacy.support.enabled`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.legacy.support.enable`, introduced in Java Agent 5.5.2

Legacy User Agent List

☞ *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

List of header values that identify legacy browsers. Entries can use the wildcard character, `*`.

Default: Empty

Type: List of strings

Hot-swap: Yes

Property: `org.forgerock.agents.legacy.user.agent.list`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.legacy.user.agent`, introduced in Java Agent 5.5.2

Legacy User Agent Redirect URI

☞ *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

The URI the Java agent uses to redirect legacy user agent requests.

Default: Empty

Type: List of URI strings

Hot-swap: Yes

Property: `org.forgerock.agents.legacy.redirect.uri`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.legacy.redirect.uri`, introduced in Java Agent 5.5.2

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.

- "Client Identification Properties"
- "Web Service Processing Properties"
- "Agent Properties"
- "JBoss Application Server Properties"
- "Cross-Site Scripting Detection Properties"
- "Fragment Relay"
- "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.

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`.

Use the properties in this section for proxies and load balancers that support provision of client IP and host name in HTTP headers.

Client IP Address Header

The name of the HTTP header that is used to determine the client IP address. If the value is not defined in the HTTP headers, the value returned by `HttpServletRequest.getRemoteAddr` is used.

Default: Empty

Type: String

Property: `org.forgerock.agents.http.header.containing.ip.address`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.client.ip.header`, introduced in Java Agent 5.5.2

Client Hostname Header

The name of the HTTP header that is used to determine the client remote host. If the value is not defined in the HTTP headers, the value returned by `HttpServletRequest.getRemoteHost` is used.

Default: Empty

Type: String

Property: `org.forgerock.agents.http.header.containing.remote.hostname`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.client.hostname.header`, introduced in Java Agent 5.5.2

Web Service Processing Properties

☞ *The following properties do not apply to Java Agents 5.7, although they may appear in the AM console:*

~~Web Service Enable~~

☞ *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

Property: `com.sun.identity.agents.config.webservice.enable`

~~Web Service End Points~~

☞ *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

Property: `com.sun.identity.agents.config.webservice.endpoint`

~~Web Service Process GET Enable~~

☞ *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

Property: `com.sun.identity.agents.config.webservice.process.get.enable`

~~Web Service Authenticator~~

☞ *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

Property: `com.sun.identity.agents.config.webservice.authenticator`

~~Web Service Response Processor~~

☞ *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

Property: `com.sun.identity.agents.config.webservice.responseprocessor`

Web Service Internal Error Content File

☞ *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

Property: `com.sun.identity.agents.config.webservice.internalerror.content`

Web Service Authorization Error Content File

☞ *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

Property: `com.sun.identity.agents.config.webservice.autherror.content`

Agent Properties

Agent Host

The host name of the Java agent.

Default: Set by installer

Type: String

Hot-swap: No

Property: `org.forgerock.agents.agent.hostname`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.agent.host`, introduced in Java Agent 5.5.2

Alternative Agent Port Number

The port number of the Java agent.

Default: Set by installer

Type: Integer

Hot-swap: No

Property: `org.forgerock.agents.agent.port`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.agent.port`, introduced in Java Agent 5.5.2

Agent Protocol

The protocol used by the Java agent.

Default: Set by installer

Type: Constrained string; probably `http` or `https`.

Hot-swap: No

Property: `org.forgerock.agents.agent.protocol`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.agent.protocol`, introduced in Java Agent 5.5.2

Agent Filter Mode

The agent filter's mode of operation. Use the following values:

- `NONE`: The Agent performs no authentication check. All requests to access resources are granted
- `SSO_ONLY`: Only users with a valid SSO token or JWT can access resources.
- `URL_POLICY`: Requests that meet requirements of an AM policy can access resources.
- `ALL`: For backward compatibility only. In a later release, this value will be changed to `URL_POLICY`.

For more information, see "Configuring the Agent Filter".

Default: `URL_POLICY`

Type: Map keyed by application name of constrained values

Property: `org.forgerock.agents.filter.mode.map`, introduced in Java Agent 5.6.2.1

Alias: `com.sun.identity.agents.config.filter.mode`, introduced in Java Agent 5.5.2

Agent Configuration Change Notification

Flag to indicate whether the Agent subscribes to WebSocket notifications from AM for configuration changes. This property only applies when you store the agent profile in AM's configuration data store.

Default: `true`

Type: Boolean

Property: `org.forgerock.agents.config.change.notifications.enabled`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.change.notification.enable`, introduced in Java Agent 5.5.2

Agent Policy Change Notification

Flag to indicate whether the Agent subscribes to WebSocket notifications from AM for policy changes.

Default: `true`

Type: Boolean

Property: `org.forgerock.agents.policy.change.notifications.enabled`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.notification.enabled`, introduced in Java Agent 5.5.2

Session Logout Notification

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

Flag to indicate whether the Agent subscribes to WebSocket notifications from AM for session logout.

Default: `true`

Type: Boolean

Property: `org.forgerock.agents.session.change.notifications.enabled`, introduced in Java Agent 5.5.2

Obsolete Agent Logout Notification

⚠ *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

Flag to indicate whether the Agent subscribes to WebSocket notifications from AM for session logout.

Use `Session Logout Notification` instead of this property.

Default: Empty

Type: Boolean

Property: `com.iplanet.am.session.client.polling.enabled`, first deprecated in Java Agent 5.5.2

Agent Notification URL

⚠ *This property does not apply to Java Agents 5.7, although it might 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.

Use `WebSocket notifications` instead of this property.

Default: Empty

Type: String; a dummy endpoint within the agent

Property: `org.forgerock.agents.obsolete.notification.url`, first deprecated in Java Agent 5.5.2

Alias: `com.sun.identity.client.notification.url`, first deprecated in Java Agent 5.5.2

JBoss Application Server Properties

WebAuthentication Available

🔔 *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

Property: `com.sun.identity.agents.config.jboss.webauth.available`

Cross-Site Scripting Detection Properties

Possible XSS code elements

Strings that, when found in the request, cause the agent to redirect the client to an error page.

Default: Empty

Type: List of cross site scripting elements

Hot-swap: Yes

Property: `org.forgerock.agents.xss.code.element.list`, introduced in Java Agent 5.7

Alias: `com.sun.identity.agents.config.xss.code.elements`, introduced in Java Agent 5.5.2

XSS detection redirect URI

Maps applications to URIs of customized pages. When a cross site scripting attack is detected, the agent redirects clients to the URI.

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: `/agentapp/XSSCodeDetected.html`

Type: Map of *application name:URI*

Hot-swap: Yes

Property: `org.forgerock.agents.xss.redirect.uri.map`, introduced in Java Agent 5.7

Alias: `com.sun.identity.agents.config.xss.redirect.uri`, introduced in Java Agent 5.5.2

Fragment Relay

Fragment Relay URI

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

A valid URI to act as a dummy endpoint for processing captured URL fragments.

When empty, unauthenticated requests to a URL with a fragment are authenticated and then redirected to the URL without the fragment. For example, a request to `http://my.domain.com:8080/myapp/index.html#chapter-1` is authenticated and redirected to `http://my.domain.com:8080/myapp/index.html`. The fragment `#chapter-1` is lost.

When set, unauthenticated requests are authenticated and then redirected to the requested URL. For example, a request to `http://my.domain.com:8080/myapp/index.html#chapter-1` is authenticated and redirected to the same URL. The fragment is not lost.

When this property is set, an extra redirect is incurred for all unauthenticated requests, to capture and process the URL fragment.

Use a dummy URI within the Agent application, such as `/agentapp/pre-authn-fragment-capture`. Avoid dummy URIs used for other purposes, such as CDSSO Redirect URI and Authentication Exchange URI.

Default: Empty

Type: String (URL of a dummy endpoint within the Agent)

Hot-swap: Yes

Property: `org.forgerock.agents.authn.fragment.relay.uri`, introduced in Java Agent 5.7

POST Data Preservation Properties

For more information, see "POST Data Preservation".

POST Data Preservation Enabled

When `true`, HTTP POST data preservation is enabled.

POST data is stored 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`

Type: Boolean

Hot-swap: Yes

Property: `org.forgerock.agents.post.data.preservation.enabled`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.postdata.preserve.enable`, introduced in Java Agent 5.5.2

Missing PDP Entry URL

A map of URLs to which the agent redirects when the PDP cache entry is discarded because of a cache timeout. The URL is expected to be a page to explain what has happened.

The Java agent redirects to URL in this list, or gives an HTTP 403 Forbidden.

Default: Emptory

Type: Map, with format *application name:URL*

Hot swap: Yes

Property: `org.forgerock.agents.pdp.noentry.url.map`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.postdata.preserve.cache.noentry.url`, introduced in Java Agent 5.5.2

PDP Cache TTL in Minutes

+ *Not available in the console for AM 6.5.x and earlier versions.*

To set as a custom property in AM, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > Advanced > Custom Properties.

The time in minutes after which entries in the Post Data Preservation cache timeout and are purged.

Default: 5

Type: Integer

Hot swap: No

Property: `org.forgerock.agents.pdp.cache.ttl.minutes`, introduced in Java Agent 5.5.2

PDP-entry TTL Milliseconds

🔗 *This property does not apply to Java Agents 5.7, although it might appear in the AM console.*

The time in milliseconds after which entries in the Post Data Preservation cache timeout and are purged. Deprecated, use PDP Cache TTL in Minutes instead.

Default: 300000

Type: Integer

Hot swap: No

Property: `com.sun.identity.agents.config.postdata.preserve.cache.entry.ttl`, first deprecated in Java Agent 5.5.2

PDP Sticky Session Mode

Specifies how to send a sticky session value, as follows:

- **URL**: Send as an HTTP parameter in the URL string.
- **Cookie**: Send as a cookie.

Default: **URL**

Type: Constrained string

Hot-swap: Yes

Property: `org.forgerock.agents.pdp.sticky.session.mode`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.postdata.preserve.stickysession.mode`, introduced in Java Agent 5.5.2

PDP Sticky Session Value

Specifies the key-value pair for sticky session mode, depending on the value of PDP Sticky Session Mode, as follows:

- When PDP Sticky Session Mode is **URL**, this property sets the name of the query parameter and its value.

For example, the value `lb=myserver` adds `lb=myserver` to the URL query string

- When PDP Sticky Session Mode is **Cookie**, this property sets the name and value for the cookie.

For example, the value `lb=myserver` sets an `lb` cookie with the value `myserver`.

Default: Empty

Type: String, of the form `name=value`

Hot-swap: Yes

Property: `org.forgerock.agents.pdp.sticky.session.value`, introduced in Java Agent 5.5.2

Alias: `com.sun.identity.agents.config.postdata.preserve.stickysession.value`, introduced in Java Agent 5.5.2

PDP Cache Total Size

The maximum number of megabytes allowed for the PDP cache. When the cache reaches the maximum, old entries are discarded.

Use this property to mitigate the risk of DoS attacks. If a malicious user posts a large amount of information to the cache, the cache size is limited.

Takes precedence if `PDP Cache Maximum Entries` is also set.

Default: Empty

Type: Integer

Hot-swap: No

Property: `org.forgerock.agents.pdp.cache.total.size.mb`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.openam.agents.config.postdata.preserve.cache.entry.max.total.size.mb`, introduced in Java Agent 5.5.2

PDP Cache Maximum Entries

The maximum number of megabytes entries in the PDP cache. When the cache reaches the maximum, old entries are discarded.

Use this property to mitigate the risk of DoS attacks. If a malicious user posts a large amount of information to the cache, the cache size is limited.

Ignored if `PDP Cache Total Size` is also set.

Default: 1000

Type: Integer

Hot-swap: No

Property: `org.forgerock.agents.pdp.cache.size`, introduced in Java Agent 5.5.2

Alias: `org.forgerock.openam.agents.config.postdata.preserve.cache.entry.max.entries`, introduced in Java Agent 5.5.2

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
<code>_id</code>	The metric ID.
<code>_type</code>	The metric type.
<code>count</code>	The number of events recorded for this metric.
<code>total</code>	The sum of the durations recorded for this metric.
<code>min</code>	The minimum duration recorded for this metric.
<code>max</code>	The maximum duration recorded for this metric.
<code>mean</code>	The mean average duration recorded for this metric.
<code>stddev</code>	The standard deviation of durations recorded for this metric.
<code>duration_units</code>	The units used for measuring the durations in the metric.
<code>p50</code>	50% of the durations recorded are at or below this value.
<code>p75</code>	75% of the durations recorded are at or below this value.
<code>p95</code>	95% of the durations recorded are at or below this value.
<code>p98</code>	98% of the durations recorded are at or below this value.
<code>p99</code>	99% of the durations recorded are at or below this value.
<code>p999</code>	99.9% of the durations recorded are at or below this value.
<code>m1_rate</code>	The one-minute average rate.
<code>m5_rate</code>	The five-minute average rate.
<code>m15_rate</code>	The fifteen-minute average rate.
<code>mean_rate</code>	The average rate.
<code>rate_units</code>	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
<code># TYPE</code>	The metric ID, and type. Note that the <code>Timer</code> metric type is reported as a <code>Summary</code> type. Formatted as a comment.
<code>_count</code>	The number of events recorded.
<code>_total</code>	The sum of the durations recorded.
<code>{quantile="0.5"}</code>	50% of the durations are at or below this value.
<code>{quantile="0.75"}</code>	75% of the durations are at or below this value.
<code>{quantile="0.95"}</code>	95% of the durations are at or below this value.
<code>{quantile="0.98"}</code>	98% of the durations are at or below this value.

Field	Description
<code>{quantile="0.99"}</code>	99% of the durations are at or below this value.
<code>{quantile="0.999"}</code>	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",}
0.01300000000000000001
ja_requests_seconds{access="granted",decision="allowed-by-policy",quantile="0.75",}
0.02200000000000000002
ja_requests_seconds{access="granted",decision="allowed-by-policy",quantile="0.95",}
0.02200000000000000002
ja_requests_seconds{access="granted",decision="allowed-by-policy",quantile="0.98",}
0.02200000000000000002
ja_requests_seconds{access="granted",decision="allowed-by-policy",quantile="0.99",}
0.02200000000000000002
ja_requests_seconds{access="granted",decision="allowed-by-policy",quantile="0.999",}
1.138000000000000001
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
<code>_id</code>	The metric ID.
<code>_type</code>	The metric type.
<code>value</code>	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
<code># TYPE</code>	The metric ID, and type. Formatted as a comment.
<code>{Metric ID}</code>	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
<code>_id</code>	The metric ID.
<code>_type</code>	The metric type. Note that the <code>distinctCounter</code> type is reported as a <code>gauge</code> type. The output formats are identical.
<code>value</code>	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
<code># TYPE</code>	The metric ID, and type. Note that the <code>distinctCounter</code> type is reported as a <code>gauge</code> type. The output formats are identical. Formatted as a comment.
<code>{Metric ID}</code>	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
<code>jvm.available-cpus</code>	Number of processors available to the Java virtual machine. (Gauge)
<code>jvm.class-loading.loaded</code>	Number of classes loaded since the Java virtual machine started. (Gauge)
<code>jvm.class-loading.unloaded</code>	Number of classes unloaded since the Java virtual machine started. (Gauge)
<code>jvm.garbage-collector.PS-MarkSweep.count</code>	Number of collections performed by the "parallel scavenge mark sweep" garbage collection algorithm. (Gauge)
<code>jvm.garbage-collector.PS-MarkSweep.time</code>	Approximate accumulated time taken by the "parallel scavenge mark sweep" garbage collection algorithm. (Gauge)

Name	Description
<code>jvm.garbage-collector.PS-Scavenge.count</code>	Number of collections performed by the "parallel scavenge" garbage collection algorithm. (Gauge)
<code>jvm.garbage-collector.PS-Scavenge.time</code>	Approximate accumulated time taken by the "parallel scavenge" garbage collection algorithm. (Gauge)
<code>jvm.memory-usage.heap.init</code>	Amount of heap memory that the Java virtual machine initially requested from the operating system. (Gauge)
<code>jvm.memory-usage.heap.max</code>	Maximum amount of heap memory that the Java virtual machine will attempt to use. (Gauge)
<code>jvm.memory-usage.heap.committed</code>	Amount of heap memory that is committed for the Java virtual machine to use. (Gauge)
<code>jvm.memory-usage.heap.used</code>	Amount of heap memory used by the Java virtual machine. (Gauge)
<code>jvm.memory-usage.total.init</code>	Amount of memory that the Java virtual machine initially requested from the operating system. (Gauge)
<code>jvm.memory-usage.total.max</code>	Maximum amount of memory that the Java virtual machine will attempt to use. (Gauge)
<code>jvm.memory-usage.non-heap.init</code>	Amount of non-heap memory that the Java virtual machine initially requested from the operating system. (Gauge)
<code>jvm.memory-usage.non-heap.max</code>	Maximum amount of non-heap memory that the Java virtual machine will attempt to use. (Gauge)
<code>jvm.memory-usage.non-heap.committed</code>	Amount of non-heap memory that is committed for the Java virtual machine to use. (Gauge)
<code>jvm.memory-usage.non-heap.used</code>	Amount of non-heap memory used by the Java virtual machine. (Gauge)
<code>jvm.memory-usage.pools.Code-Cache.init</code>	Amount of "code cache" memory that the Java virtual machine initially requested from the operating system. (Gauge)
<code>jvm.memory-usage.pools.Code-Cache.max</code>	Maximum amount of "code cache" memory that the Java virtual machine will attempt to use. (Gauge)
<code>jvm.memory-usage.pools.Code-Cache.committed</code>	Amount of "code cache" memory that is committed for the Java virtual machine to use. (Gauge)
<code>jvm.memory-usage.pools.Code-Cache.used</code>	Amount of "code cache" memory used by the Java virtual machine. (Gauge)
<code>jvm.memory-usage.pools.Compressed-Class-Space.init</code>	Amount of "compressed class space" memory that the Java virtual machine initially requested from the operating system. (Gauge)
<code>jvm.memory-usage.pools.Compressed-Class-Space.init</code>	Maximum amount of "compressed class space" memory that the Java virtual machine will attempt to use. (Gauge)

Name	Description
<code>jvm.memory-usage.pools.Compressed-Class-Space.committed</code>	Amount of "compressed class space" memory that is committed for the Java virtual machine to use. (Gauge)
<code>jvm.memory-usage.pools.Compressed-Class-Space.used</code>	Amount of "compressed class space" memory used by the Java virtual machine. (Gauge)
<code>jvm.memory-usage.pools.Metaspace.init</code>	Amount of "metaspace" memory that the Java virtual machine initially requested from the operating system. (Gauge)
<code>jvm.memory-usage.pools.Metaspace.max</code>	Maximum amount of "metaspace" memory that the Java virtual machine will attempt to use. (Gauge)
<code>jvm.memory-usage.pools.Metaspace.committed</code>	Amount of "metaspace" memory that is committed for the Java virtual machine to use. (Gauge)
<code>jvm.memory-usage.pools.Metaspace.used</code>	Amount of "metaspace" memory used by the Java virtual machine. (Gauge)
<code>jvm.memory-usage.pools.PS-Eden-Space.init</code>	Amount of "parallel scavenge eden space" memory that the Java virtual machine initially requested from the operating system. (Gauge)
<code>jvm.memory-usage.pools.PS-Eden-Space.max</code>	Maximum amount of "parallel scavenge eden space" memory that the Java virtual machine will attempt to use. (Gauge)
<code>jvm.memory-usage.pools.PS-Eden-Space.committed</code>	Amount of "parallel scavenge eden space" memory that is committed for the Java virtual machine to use. (Gauge)
<code>jvm.memory-usage.pools.PS-Eden-Space.used-after-gc</code>	Amount of "parallel scavenge eden space" memory after the last time garbage collection recycled unused objects in this memory pool. (Gauge)
<code>jvm.memory-usage.pools.PS-Eden-Space.used</code>	Amount of "parallel scavenge eden space" memory used by the Java virtual machine. (Gauge)
<code>jvm.memory-usage.pools.PS-Old-Gen.init</code>	Amount of "parallel scavenge old generation" memory that the Java virtual machine initially requested from the operating system. (Gauge)
<code>jvm.memory-usage.pools.PS-Old-Gen.max</code>	Maximum amount of "parallel scavenge old generation" memory that the Java virtual machine will attempt to use. (Gauge)
<code>jvm.memory-usage.pools.PS-Old-Gen.committed</code>	Amount of "parallel scavenge old generation" memory that is committed for the Java virtual machine to use. (Gauge)
<code>jvm.memory-usage.pools.PS-Old-Gen.used-after-gc</code>	Amount of "parallel scavenge old generation" memory after the last time garbage collection recycled unused objects in this memory pool. (Gauge)
<code>jvm.memory-usage.pools.PS-Old-Gen.used</code>	Amount of "parallel scavenge old generation" memory used by the Java virtual machine. (Gauge)

Name	Description
<code>jvm.memory-usage.pools.PS-Survivor-Space.init</code>	Amount of "parallel scavenge survivor space" memory that the Java virtual machine initially requested from the operating system. (Gauge)
<code>jvm.memory-usage.pools.PS-Survivor-Space.max</code>	Maximum amount of "parallel scavenge survivor space" memory that the Java virtual machine will attempt to use. (Gauge)
<code>jvm.memory-usage.pools.PS-Survivor-Space.committed</code>	Amount of "parallel scavenge survivor space" memory that is committed for the Java virtual machine to use. (Gauge)
<code>jvm.memory-usage.pools.PS-Survivor-Space.used-after-gc</code>	Amount of "parallel scavenge survivor space" memory after the last time garbage collection recycled unused objects in this memory pool. (Gauge)
<code>jvm.memory-usage.pools.PS-Survivor-Space.used</code>	Amount of "parallel scavenge survivor space" memory used by the Java virtual machine. (Gauge)
<code>jvm.memory-usage.total.committed</code>	Amount of memory that is committed for the Java virtual machine to use. (Gauge)
<code>jvm.memory-usage.total.used</code>	Amount of memory used by the Java virtual machine. (Gauge)
<code>jvm.thread-state.blocked.count</code>	Number of threads in the BLOCKED state. (Gauge)
<code>jvm.thread-state.count</code>	Number of live threads including both daemon and non-daemon threads. (Gauge)
<code>jvm.thread-state.daemon.count</code>	Number of live daemon threads. (Gauge)
<code>jvm.thread-state.new.count</code>	Number of threads in the NEW state. (Gauge)
<code>jvm.thread-state.runnable.count</code>	Number of threads in the RUNNABLE state. (Gauge)
<code>jvm.thread-state.terminated.count</code>	Number of threads in the TERMINATED state. (Gauge)
<code>jvm.thread-state.timed_waiting.count</code>	Number of threads in the TIMED_WAITING state. (Gauge)
<code>jvm.thread-state.waiting.count</code>	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}
```

SSO Token to JWT Exchange Metrics

Java Agents expose the following SSO token to JWT exchange-related monitoring metrics:

sso-exchange.cache.size

The size of the SSO token exchange cache. (Gauge)

Prometheus name:

```
ja_sso_exchange_cache_size
```

sso-exchange.cache.eviction

The eviction count for the SSO token exchange cache. (Gauge)

Prometheus name:

```
ja_sso_exchange_cache_eviction
```

sso-exchange.cache.load-count

The load count for the SSO token exchange cache. (Gauge)

Prometheus name:

```
ja_sso_exchange_cache_load_count
```

sso-exchange.cache.load-time

The load time for the SSO token exchange, in milliseconds. (Gauge)

Prometheus name:

```
ja_sso_exchange_cache_load_time
```

sso-exchange.cache.hit

The hit count for the SSO token exchange cache. (Gauge)

Prometheus name:

```
ja_sso_exchange_cache{outcome=hit}
```

sso-exchange.cache.miss

The miss count for the SSO token exchange cache. (Gauge)

Prometheus name:

```
ja_sso_exchange_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.

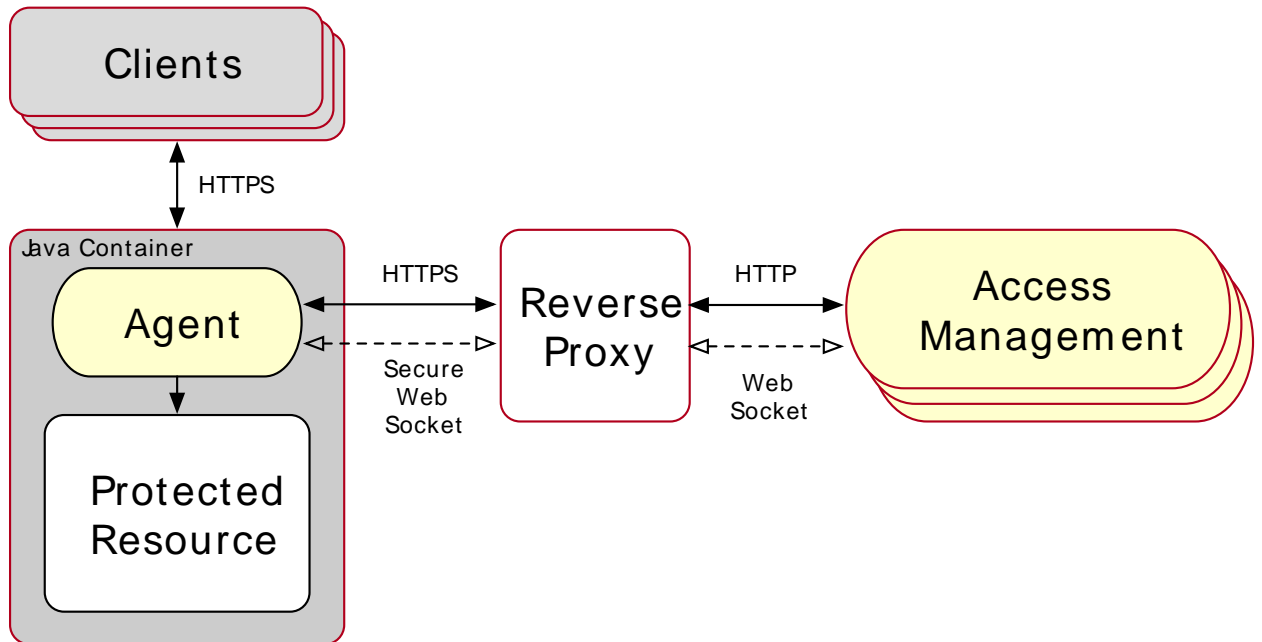
`--getEncryptKey`

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"

Implementing Custom Task Handlers

This section gives steps and example handlers for adding a custom task handler to the list of handlers the agent runs for each resource request.

1. Place `com.sun.identity.agents.arch.ServiceResolver` on the classpath.
2. Add `com.sun.identity.agents.arch.ServiceResolver` to the bootstrap property Service Resolver Class Name.

Example of a Custom Filter Result Task Handler

```
/*
 * Copyright 2019-2020 ForgeRock AS. All Rights Reserved
 *
 * Use of this code requires a commercial software license with ForgeRock AS.
 * or with one of its affiliates. All use shall be exclusively subject
 * to such license between the licensee and ForgeRock AS.
 */
package com.sun.identity.agents.custom;

import static org.forgerock.agents.debug.AgentDebug.logTrace;

import javax.servlet.http.HttpServletRequest;

import org.forgerock.agents.util.Utills;

import com.sun.identity.agents.arch.AgentConfiguration;
import com.sun.identity.agents.arch.Manager;
import com.sun.identity.agents.filter.AmFilterMode;
import com.sun.identity.agents.filter.AmFilterRequestContext;
import com.sun.identity.agents.filter.AmFilterResult;
import com.sun.identity.agents.filter.AmFilterResultHandler;

/**
 * This is an example of a custom filter result task handler
 */
@SuppressWarnings("unused")
public class CustomFilterResultTaskHandler extends AmFilterResultHandler {

    public CustomFilterResultTaskHandler(Manager manager) {
        super(manager);
    }

    @Override
    public boolean isActive() {
        return true;
    }

    @Override
    public String getHandlerName() {
        return "CustomFilterResultTaskHandler";
    }

    @Override
    public AmFilterResult process(AmFilterRequestContext context, AmFilterResult result) {
```

```

String applicationName = Utils.getApplicationName(context);
AmFilterMode amFilterMode = AgentConfiguration.getTheFilterMode(applicationName);
HttpServletRequest request = context.getHttpServletRequest();

logTrace("Hello from {}, application name {}, filter mode {}, {} {}, result {}",
    getHandlerName(), applicationName, amFilterMode,
    request.getMethod(), request.getRequestURI(),
    result.toString());

// Must return the result parameter, unless you have a really good reason not to.
return result;
}
}

```

Example of a Custom Self-Redirect Task Handler

```

/*
 * Copyright 2019-2020 ForgeRock AS. All Rights Reserved
 *
 * Use of this code requires a commercial software license with ForgeRock AS.
 * or with one of its affiliates. All use shall be exclusively subject
 * to such license between the licensee and ForgeRock AS.
 */
package com.sun.identity.agents.custom;

import static org.forgerock.agents.debug.AgentDebug.logTrace;

import javax.servlet.http.HttpServletRequest;

import org.forgerock.agents.util.Utils;

import com.sun.identity.agents.arch.AgentConfiguration;
import com.sun.identity.agents.arch.AgentException;
import com.sun.identity.agents.arch.Manager;
import com.sun.identity.agents.filter.AmFilterMode;
import com.sun.identity.agents.filter.AmFilterRequestContext;
import com.sun.identity.agents.filter.AmFilterResult;
import com.sun.identity.agents.filter.AmFilterTaskHandler;
import com.sun.identity.agents.filter.IBaseAuthnContext;

/**
 * This is an example of a custom self-redirect task handler. It is essentially the same as the inbound
 * task handler.
 */
@SuppressWarnings("unused")
public class CustomSelfRedirectTaskHandler extends AmFilterTaskHandler {

    public CustomSelfRedirectTaskHandler(Manager manager) {
        super(manager);
    }

    @Override
    public void initialize(IBaseAuthnContext context) throws AgentException {
        super.initialize(context);
    }
}

```

```

}

@Override
public boolean isActive() {
    return true;
}

@Override
public String getHandlerName() {
    return "Custom self redirect task handler";
}

@Override
public AmFilterResult process(AmFilterRequestContext context) {

    String applicationName = Utils.getApplicationName(context);
    AmFilterMode amFilterMode = AgentConfiguration.getTheFilterMode(applicationName);
    HttpServletRequest request = context.getHttpServletRequest();

    logTrace("Hello from {}, application name {}, filter mode {}, {} {}",
        getHandlerName(), applicationName, amFilterMode,
        request.getMethod(), request.getRequestURI());

    // return null to continue to the other task handlers (until one returns a non null value)
    // return AmFilterResultStatus.STATUS_CONTINUE to grant access (continue to the next filter after
the agent)
    // return AmFilterResultStatus.STATUS_REDIRECT to redirect somewhere else
    // return AmFilterResultStatus.STATUS_FORBIDDEN to deny access
    // return AmFilterResultStatus.STATUS_SERVE_DATA to serve up data to the browser
    // return AmFilterResultStatus.STATUS_SERVER_ERROR to abort the request with a 500 server error
    //
    return null;
}
}

```

Example of a Custom Inbound Task Handler

```

/*
 * Copyright 2019-2020 ForgeRock AS. All Rights Reserved
 *
 * Use of this code requires a commercial software license with ForgeRock AS.
 * or with one of its affiliates. All use shall be exclusively subject
 * to such license between the licensee and ForgeRock AS.
 */
package com.sun.identity.agents.custom;

import static org.forgerock.agents.debug.AgentDebug.LogTrace;

import javax.servlet.http.HttpServletRequest;

import org.forgerock.agents.util.Utils;

import com.sun.identity.agents.arch.AgentConfiguration;
import com.sun.identity.agents.arch.AgentException;
import com.sun.identity.agents.arch.Manager;
import com.sun.identity.agents.filter.AmFilterMode;
import com.sun.identity.agents.filter.AmFilterRequestContext;

```



```

import com.sun.identity.agents.filter.AmFilterResult;
import com.sun.identity.agents.filter.AmFilterTaskHandler;
import com.sun.identity.agents.filter.IBaseAuthnContext;

/**
 * This is an example of a custom inbound task handler
 */
@SuppressWarnings("unused")
public class CustomInboundTaskHandler extends AmFilterTaskHandler {

    public CustomInboundTaskHandler(Manager manager) {
        super(manager);
    }

    @Override
    public void initialize(IBaseAuthnContext context) throws AgentException {
        super.initialize(context);
    }

    @Override
    public boolean isActive() {
        return true;
    }

    @Override
    public String getHandlerName() {
        return "Custom inbound task handler";
    }

    @Override
    public AmFilterResult process(AmFilterRequestContext context) {

        String applicationName = Utils.getApplicationName(context);
        AmFilterMode amFilterMode = AgentConfiguration.getTheFilterMode(applicationName);
        HttpServletRequest request = context.getHttpServletRequest();

        logTrace("Hello from {}, application name {}, filter mode {}, {} {}",
            getHandlerName(), applicationName, amFilterMode,
            request.getMethod(), request.getRequestURI());

        // return null to continue to the other task handlers (until one returns a non null value)
        // return AmFilterResultStatus.STATUS_CONTINUE to grant access (continue to the next filter after
        the agent)
        // return AmFilterResultStatus.STATUS_REDIRECT to redirect somewhere else
        // return AmFilterResultStatus.STATUS_FORBIDDEN to deny access
        // return AmFilterResultStatus.STATUS_SERVE_DATA to serve up data to the browser
        // return AmFilterResultStatus.STATUS_SERVER_ERROR to abort the request with a 500 server error
        //
        return null;
    }
}

```

Example of How to Override the ServiceResolver Class

```

/*
 * Copyright 2019-2020 ForgeRock AS. All Rights Reserved
 *

```

```
* Use of this code requires a commercial software license with ForgeRock AS.
* or with one of its affiliates. All use shall be exclusively subject
* to such license between the licensee and ForgeRock AS.
*/
package com.sun.identity.agents.custom;

import java.util.ArrayList;
import java.util.List;

import com.sun.identity.agents.arch.ServiceResolver;

/**
 * This is an example of how to override the ServiceResolver class to provide your own custom task
 * handlers. To use
 * this example class, place the following in the custom properties on the advanced tab in the Java Agents
 * profile:
 * <p></p>
 *
 * org.forgerock.agents.service.resolver.class.name=com.sun.identity.agents.custom.CustomServiceResolverExample
 * <p></p>
 * and restart the agent.
 */
@SuppressWarnings("unused")
public class CustomServiceResolverExample extends ServiceResolver {

    @Override
    public List<String> getPreInboundTaskHandlers() {
        List<String> result = new ArrayList<>();
        result.add(CustomInboundTaskHandler.class.getName());
        return result;
    }

    @Override
    public List<String> getPostInboundTaskHandlers() {
        return new ArrayList<>();
    }

    @Override
    public List<String> getPreSelfRedirectHandlers() {
        List<String> result = new ArrayList<>();
        result.add(CustomSelfRedirectTaskHandler.class.getName());
        return result;
    }

    @Override
    public List<String> getPostSelfRedirectHandlers() {
        return new ArrayList<>();
    }

    @Override
    public List<String> getPreFilterResultHandlers() {
        List<String> result = new ArrayList<>();
        result.add(CustomFilterResultTaskHandler.class.getName());
        return result;
    }

    @Override
    public List<String> getPostFilterResultHandlers() {
        return new ArrayList<>();
    }
}
```

```
}  
}
```

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	<p>In general terms, a service exposing protected resources.</p> <p>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.</p>
Application type	<p>Application types act as templates for creating policy applications.</p> <p>Application types define a preset list of actions and functional logic, such as policy lookup and resource comparator logic.</p>

	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 <i>reference</i> 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

	<p>request. For browser-based clients, AM sets a cookie in the browser that contains the session information.</p> <p>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.</p>
Conditions	<p>Defined as part of policies, these determine the circumstances under which which a policy applies.</p> <p>Environmental conditions reflect circumstances like the client IP address, time of day, how the subject authenticated, or the authentication level achieved.</p> <p>Subject conditions reflect characteristics of the subject like whether the subject authenticated, the identity of the subject, or claims in the subject's JWT.</p>
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 <i>reference</i> 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	<p>Group of AM servers configured the same way, accessed through a load balancer layer. The load balancer handles failover to provide service-level availability.</p> <p>The load balancer can also be used to protect AM services.</p>
Standard metadata	Standard federation configuration information that you can share with other access management software.
Stateless Service	<p>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.</p> <p>All AM services are stateless unless otherwise specified. See also Client-based sessions and CTS-based sessions.</p>
Subject	<p>Entity that requests access to a resource</p> <p>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.</p>
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.