

# User Guide

## ON THIS PAGE

User Guide

About ForgeRock Identity Platform™ Software

About Java Agent

Agent Components

Agent Configuration

Request Flow

Features

Cross-Domain Single Sign-On

Login Redirect

Not Enforced Rules

POST Data Preservation

Continuous Security

Attribute Fetch Modes

Autonomous Mode

Login Attempt Limits

FQDN Checking

Cookie Reset Properties

Caching

Query Parameter Handling

Authentication Failure

Connection Pooling

Configuration for Load Balancers and Reverse Proxies

Identifying Clients Behind Load Balancers and Reverse Proxies

Agent - Load Balancer/Reverse Proxy - AM

Agent - Load Balancer/Reverse Proxy - Client

Troubleshooting

Questions and Answers

Reference

agentadmin Command

Configure Apache HTTP Server As a Reverse Proxy Example

Glossary

## User Guide

---

This guide describes how to use ForgeRock Access Management Java Agent.

### 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>.

### About Java Agent

---

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

Java Agents intercept inbound requests to web 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 web applications.

### Agent Components

Java Agent includes the following main components:

#### **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 part of Java Agent, but they play an important part in the agent operation:

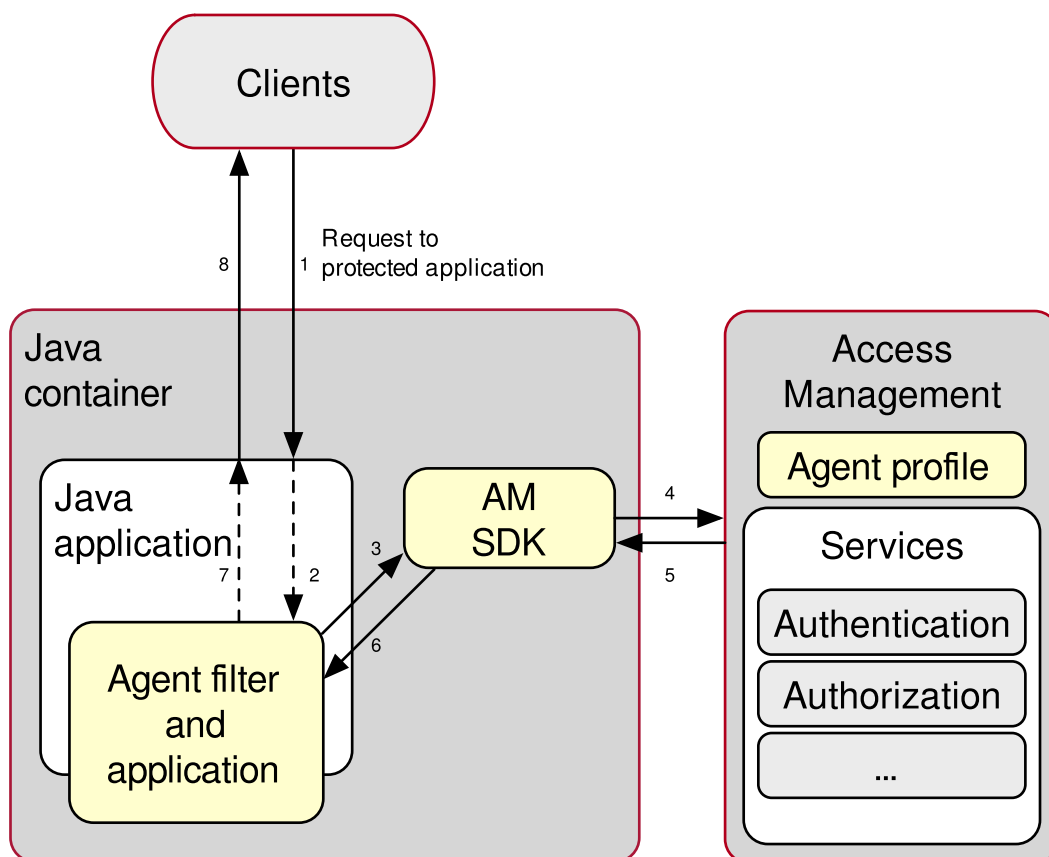
### AM SDKs

A set of APIs required to interact with AM.

### Agent Profile

A set of configuration properties that define the agent behavior. The agent profile can be stored in AM's configuration store or as a text file local to the agent installation.

The following image shows the Java Agent components when the agent profile is stored in the AM configuration store:



## Agent Configuration

Java Agent uses the configuration files described in this section.

The files must be in a location defined by a property added to `JAVA_OPTS`. For example, in Tomcat, the agent can take the file location from `bin/setenv.sh`, as follows:

```
JAVA_OPTS="$JAVA_OPTS -  
Dopenam.agents.bootstrap.dir=/path/to/agents/agent/agent_instance
```

```
/config"
```

## *AgentBootstrap.properties*

This file defines bootstrap parameters. The following information is required in the file:

- Private AM URL:

Used for communication with AM, for example, to retrieve policy information or user information. The URL is assembled from the following properties, and is required, even if the agent never contacts AM:

- AM Authentication Service Protocol
- AM Authentication Service Host Name
- AM Authentication Service Port
- AM Authentication Service Path

- Public AM URL:

This URL must be provided by the user if the AM firewall rules distinguish between public and private URL. The agent uses this property to redirect the user's browser to public-facing URLs for login. If it is not provided, the AM private URL is used.

- Agent Profile:

- Agent Profile Name
- Agent Profile Realm

- Location of Agent Configuration Repository:

Defines the agent configuration mode:

- Local configuration mode

The agent reads its configuration from `AgentConfiguration.properties`. When the agent is in this mode, it ignores changes made to the agent profile in AM.

Depending on the configuration in the `AgentConfiguration.properties` file, the agent might never need to contact AM. For example, if Autonomous mode is `true`, the agent runs independently of an AM instance.

- Remote configuration mode (default)

The agent ignores the configuration in `AgentConfiguration.properties`, retains the retrieved bootstrap properties, and downloads the configuration from AM.

When the first user request is made, the agent contacts AM to retrieve the agent configuration. If AM is unavailable, the request returns an HTTP 403

Permission denied.

### *AgentConfiguration.properties*

This file defines agent configuration settings, and overrides any properties set in the bootstrap file.

If the value of Location of Agent Configuration Repository is LOCAL , the agent reads the AgentConfiguration.properties file after AgentBootstrap.properties . If the value of Location of Agent Configuration Repository is REMOTE , the agent ignores this file.

In Java Agent 5.7 and earlier versions, this file was named OpenSSOAgentConfiguration.properties .

### *AgentPassword.properties*

This file defines an encrypted password for the agent profile. For more information, see Encrypted Agent Password.

### *AgentKey.properties*

This file defines the key used to encrypt the agent password. For more information, see Encryption Key.

## *Changing the Agent Configuration*

Change the agent configuration in the following ways:

### ***Change the agent bootstrap configuration***

Manually edit AgentBootstrap.properties , and then restart the container running the agent.

### ***Change the agent configuration in LOCAL mode***

Manually edit the AgentConfiguration.properties file, and set a value for Configuration Reload Interval.

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

The value of Location of Agent Configuration Repository must be LOCAL .

### ***Change the agent configuration in 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.

The value of Location of Agent Configuration Repository must be REMOTE .

## Change the agent configuration on the AM console

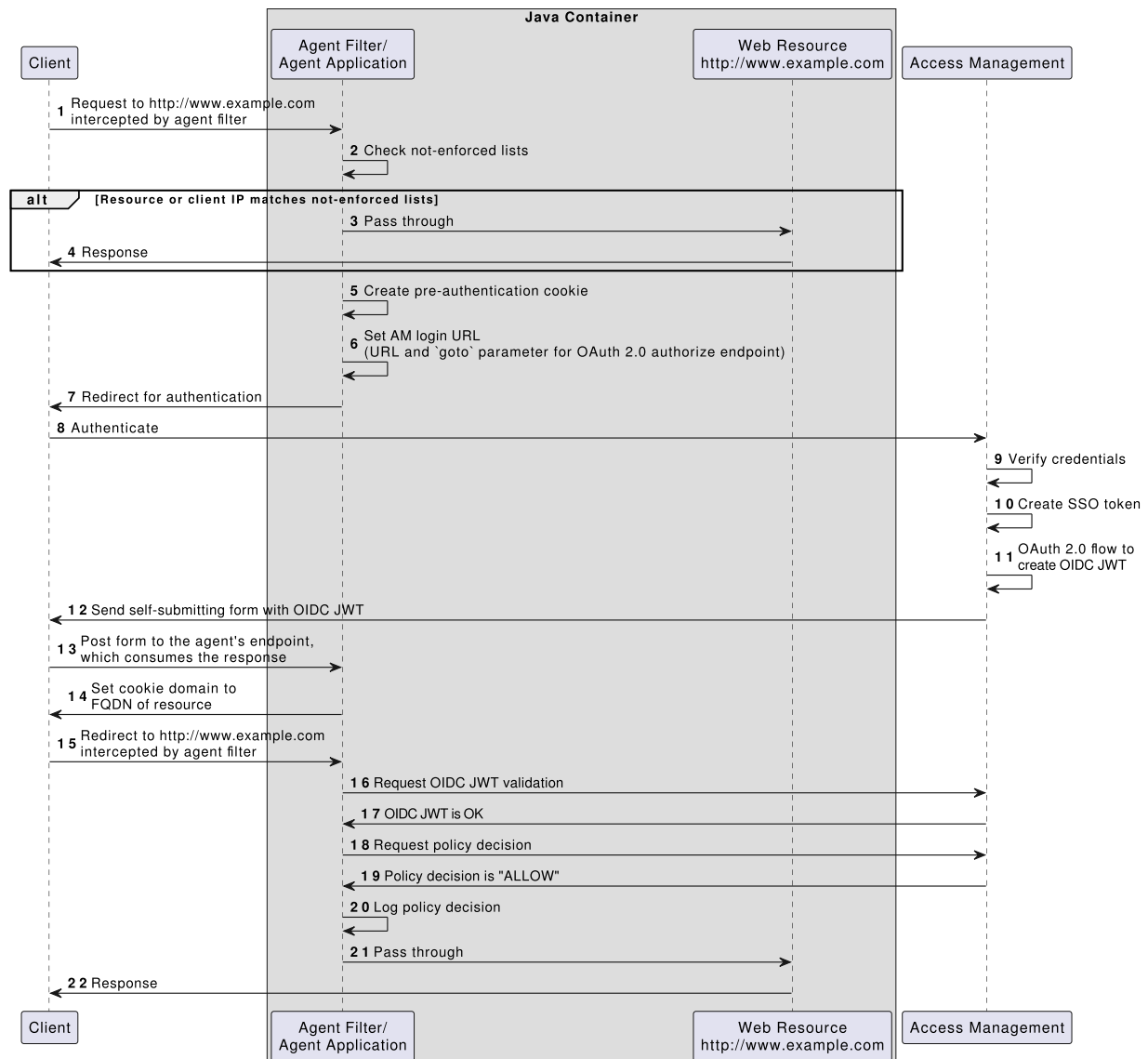
Go to REALMS > **Realm Name** > Applications > **agents** > Java > **agent name**.

The value of Location of Agent Configuration Repository must be REMOTE .

## Request Flow

When a client requests access to an application resource, the Java Agent intercepts the request. AM then validates the identity of the client, and their authorization to access the protected resource.

The following simplified data flow occurs when an unauthenticated client requests a resource protected by a Java Agent and AM. The flow assumes that requests must meet the requirements of an AM policy. For a detailed diagram, see [Single Sign-On](#) in AM's *Authentication and Single Sign-On Guide*.



1 An unauthenticated client attempts to access a resource, and the agent intercepts the inbound request.

- 2 The agent evaluates whether the requested resource or the client IP address matches a not-enforced rule.
- 3-4 *Alternate Flow*. The requested resource or the client IP address matches a not-enforced rule. The agent allows access to the resource, and the client receives a response from `www.example.com`. The flow ends.
- 5 The agent creates a pre-authentication cookie to protect against reply attacks. 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.
- 6 The agent sets the AM login URL, which includes the `goto` parameter for the OAuth 2.0 authorize endpoint, and
- 7 The agent redirects the client to log in to AM.
- 8-12 The client authenticates to AM:
- AM's Authentication Service verifies the client credentials. AM creates an SSO token, and OIDC JWT with session information.
  - AM sends the client a self-submitting form with the OIDC JWT.
- 13 The client posts the self-submitting form to the agent endpoint, and the Java Agent consumes it.
- 14 The agent sets the cookie domain to the FQDN of the resource.
- 15 The client attempts to access the protected resource again, and the agent intercepts the request.
- 16 The agent contacts AM to validate the session contained in the OIDC JWT.
- 17 AM validates the session.
- 18 The agent contacts AM's Policy Service, requesting a decision about whether the client is authorized to access the resource.
- 19 AM's Policy Service returns `ALLOW`.
- 20 The agent writes the policy decision to the audit log.
- 21 The agent enforces the policy decision. Because the Policy Service returned `ALLOW`, the agent performs a pass-through operation to return the resource to the client.
- 22 The client accesses the resource.

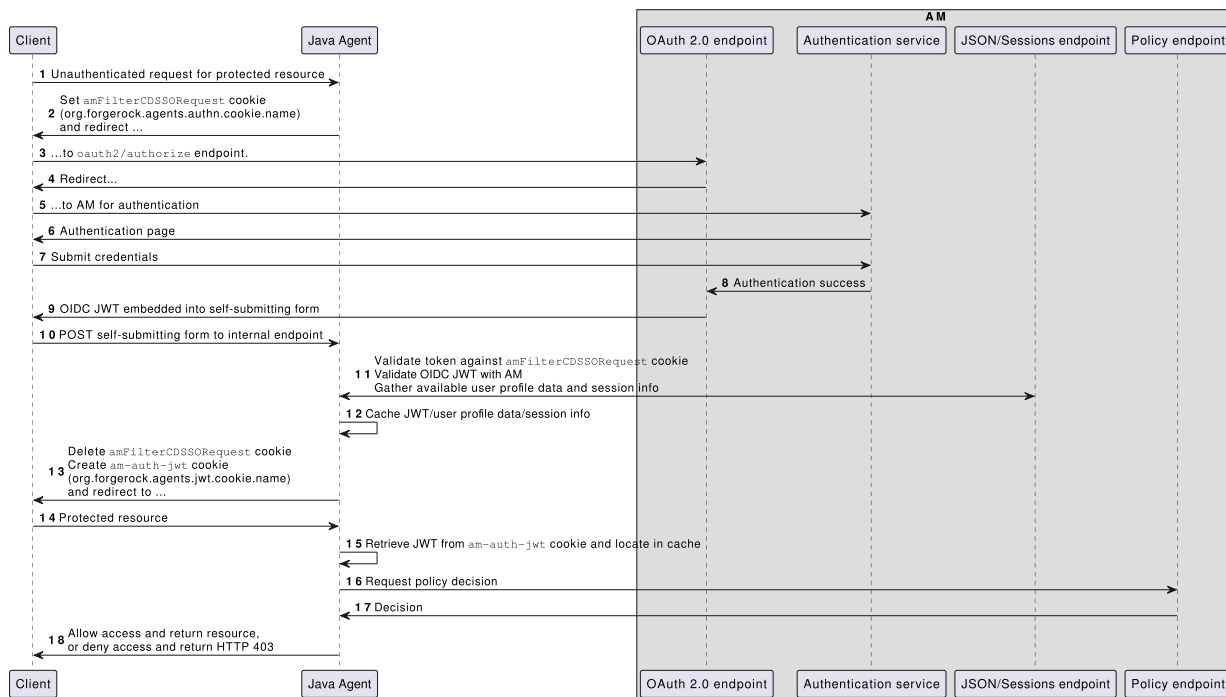
## Features

---

# Cross-Domain Single Sign-On

In cross-domain single sign-on, Java Agent processes requests using authentication provided by AM. Users can access multiple independent services from a single login session, using the agent to transfer the session ID. The agent and AM can be in the same domain or in different domains.

The following diagram illustrates the CDSSO flow:



When the agent is in local configuration mode, configure the Authentication Redirect URI. When the agent is in remote configuration mode, the value is set by the agent configuration in AM.

For more information, see Single Sign-On and Implementing CDSSO in AM's *Authentication and Single Sign-On Guide*.

## Login Redirect

When an unauthenticated user requests access to a protected resource, the agent redirects the user to log in. The login redirect can be to a specific AM instance, an AM site, or a website. For example, the agent can redirect a user from the `france.example.com` domain to log in to the `am.france.example.com` AM site.

This section describes a default login redirect, and options to configure the redirect.

### Default Login Redirect

The following image shows the flow of data during a default login redirect, when an unauthenticated user requests access to a protected resource. The agent wraps the SSO



session token inside an OIDC JWT. Authentication requires access to `XUI/*`, and the agent requires access to `json/authenticate`, `json/sessions`, and `json/serverinfo`.

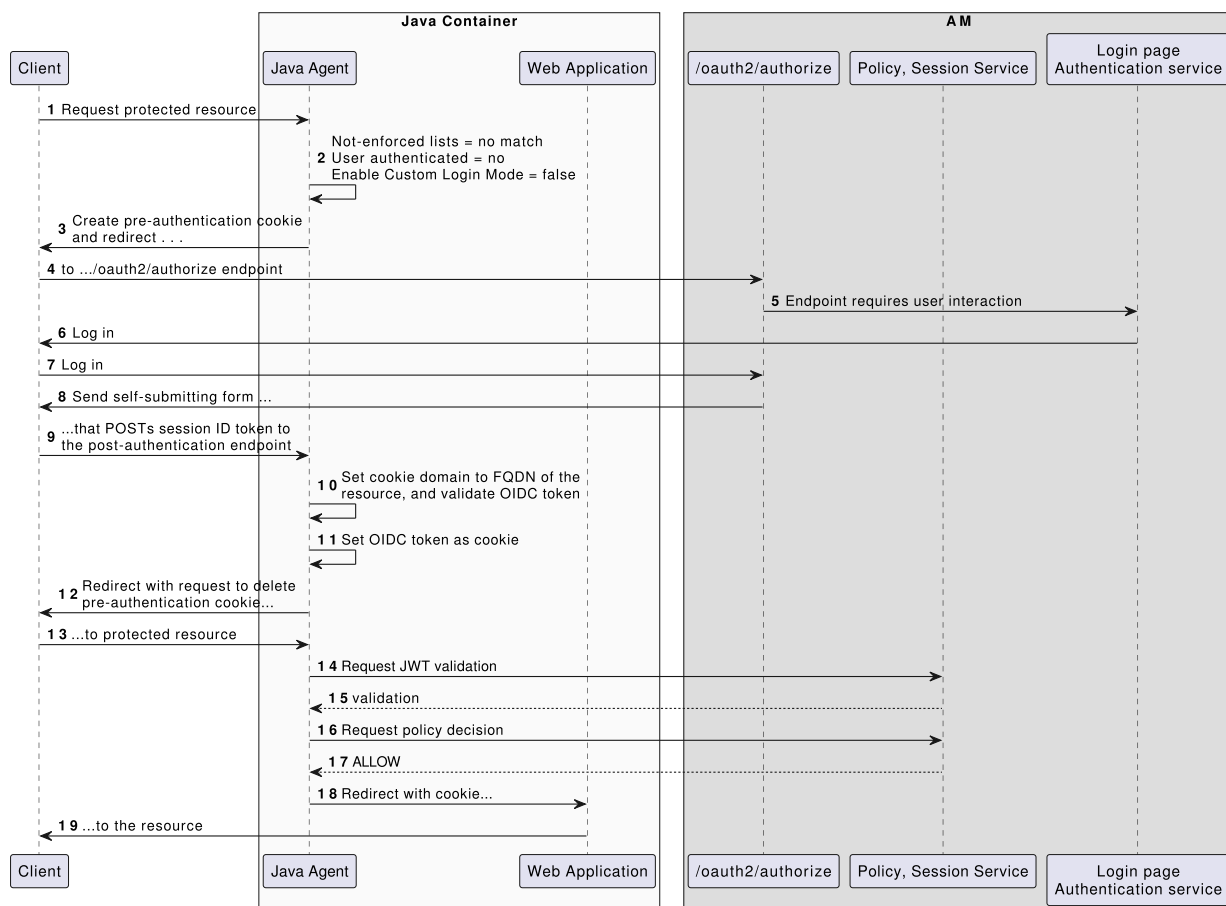


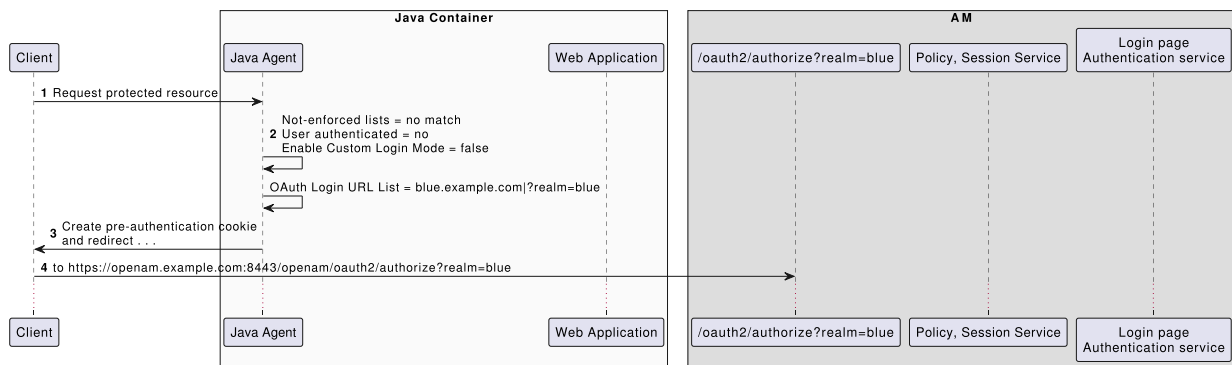
Figure 1. Data Flow for Default Login Redirect

### Redirect Login to a Realm Based on the Request Domain

Set the following properties to redirect a request to a login realm, based on the request domain:

- Enable Custom Login Mode: Leave with the default value of `false`.
- OAuth Login URL List: Map the request domain to the required login realm. When this property is set, the agent tries to match the request domain to the list of domains in this property. If there is a match, the agent redirects the user to log in at the matched URI.

The following image shows the first few steps from figure 1, where the redirect is this time configured with the value in OAuth Login URL List.



In the following example, unauthenticated requests are redirected to different login realms according to the request domain:

```
org.forgerock.agents.oauth.login.url.list[0] = blue.example.com|?realm=blue
org.forgerock.agents.oauth.login.url.list[1] = red.example.com|?realm=red
org.forgerock.agents.oauth.login.url.list[2] = red.example.com/yellow|?realm=orange
org.forgerock.agents.oauth.login.url.list[3] = |?realm=default
```

The requests are directed as follows:

- Requests for a resource in `blue.example.com` are passed to the `oauth2/authorize` endpoint to log the user into the blue realm.
- Requests for a resource in `red.example.com/ruby` are passed to the `oauth2/authorize` endpoint to log the user into the red realm.
- Requests for a resource in `red.example.com/yellow/` are passed to the `oauth2/authorize` endpoint to log the user into the orange realm.
- Requests for a resource in an unmapped domain are passed to the `oauth2/authorize` endpoint to log the user in to the specified default realm.

### *Redirect Login to an OIDC Endpoint Based on the Request Domain*

Set the following properties to redirect a request to a different OIDC endpoint, based on the request domain:

- **Enable Custom Login Mode:** Leave with the default value of `false`.
- **OAuth Login URL List:** Map the request domain to the required login endpoint. When this property is set, the agent tries to match the request domain to the list of domains in this property. If there is a match, the agent redirects the user to the matched URI for login.

In the following example, unauthenticated requests are redirected to different OIDC endpoints according to the request domain:

```
org.forgerock.agents.oauth.login.url.list[2] =
red.example.com/yellow|https://other.example.com:8081/am/other-
idp/oauth2/other-authorize?realm=orange
org.forgerock.agents.oauth.login.url.list[3] = |?realm=default
```

The requests are directed as follows:

- Requests for a resource in `red.example.com/yellow`, are passed to `https://other.example.com:8081/am/other-idp/oauth2/other-authorize` to log the user in to the orange realm.
- Requests for a resource in an unmapped domain are passed to the `oauth2/authorize` endpoint to log the user in to the specified default realm.

### *Redirect Login for Legacy Deployments That Require SSO Tokens*

For legacy deployments that require SSO tokens, redirect the login to a custom login page to set the SSO token in the user's browser after authentication. 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.
- Users cannot access the AM servers directly in your environment.
- Custom login pages are not part of AM's XUI.

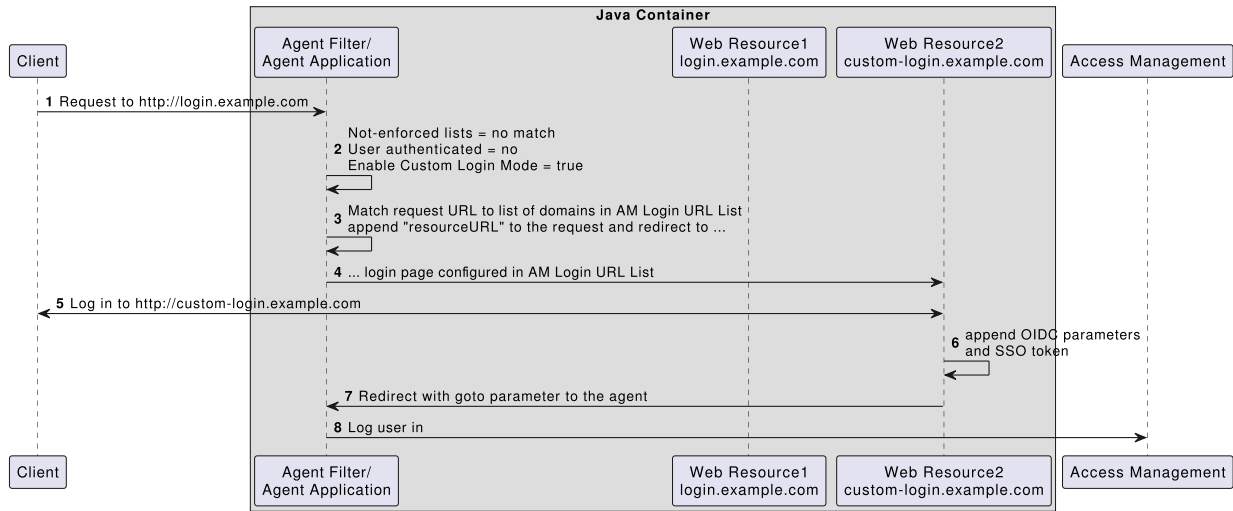
Use the following properties to configure custom login redirects:

- Enable Custom Login Mode:
  - Set to `true` if the custom login page sets an SSO token in the user's browser after authentication.
- Enable SSO Token Acceptance:
  - Set to `true` to prevent the agent from attempting to convert SSO tokens into OIDC tokens. Use this option to continue to use SSO tokens, when the agent and the token issuer are in the same domain.
  - Set to `false` to allow the agent to convert SSO tokens to OIDC tokens. Use this option for better security, and in web applications and APIs where the backend requires user information in form of an OIDC token.
- AM Login URL List or Legacy Login URL List
  - If AM Login URL List is configured, the agent redirects the user to the specified custom login page, and appends a `resourceURL` parameter to the login endpoint.
  - If Legacy Login URL List is configured but AM Login URL List is not, the agent matches the request with the domains and URLs specified by Legacy Login URL

List, and redirects the user to the appropriate custom login page.

- If neither property is configured, authentication fails.

The following image shows a simplified flow for a custom login redirect, when the agent uses SSO tokens without converting them to OIDC id\_tokens:



### Redirect Login to Pages Behind a Firewall

When a login page is in a network that accesses AM through a proxy, firewall, or other technology that remaps the AM URL, set Public AM URL to map the AM URL to a URL that the login page can access.

For example, if AM and the agent communicate on `example-internal.com`, but the custom login pages are on `example-external.com`, configure the public AM URL as `https://openam.example-external.com:8443/openam`.

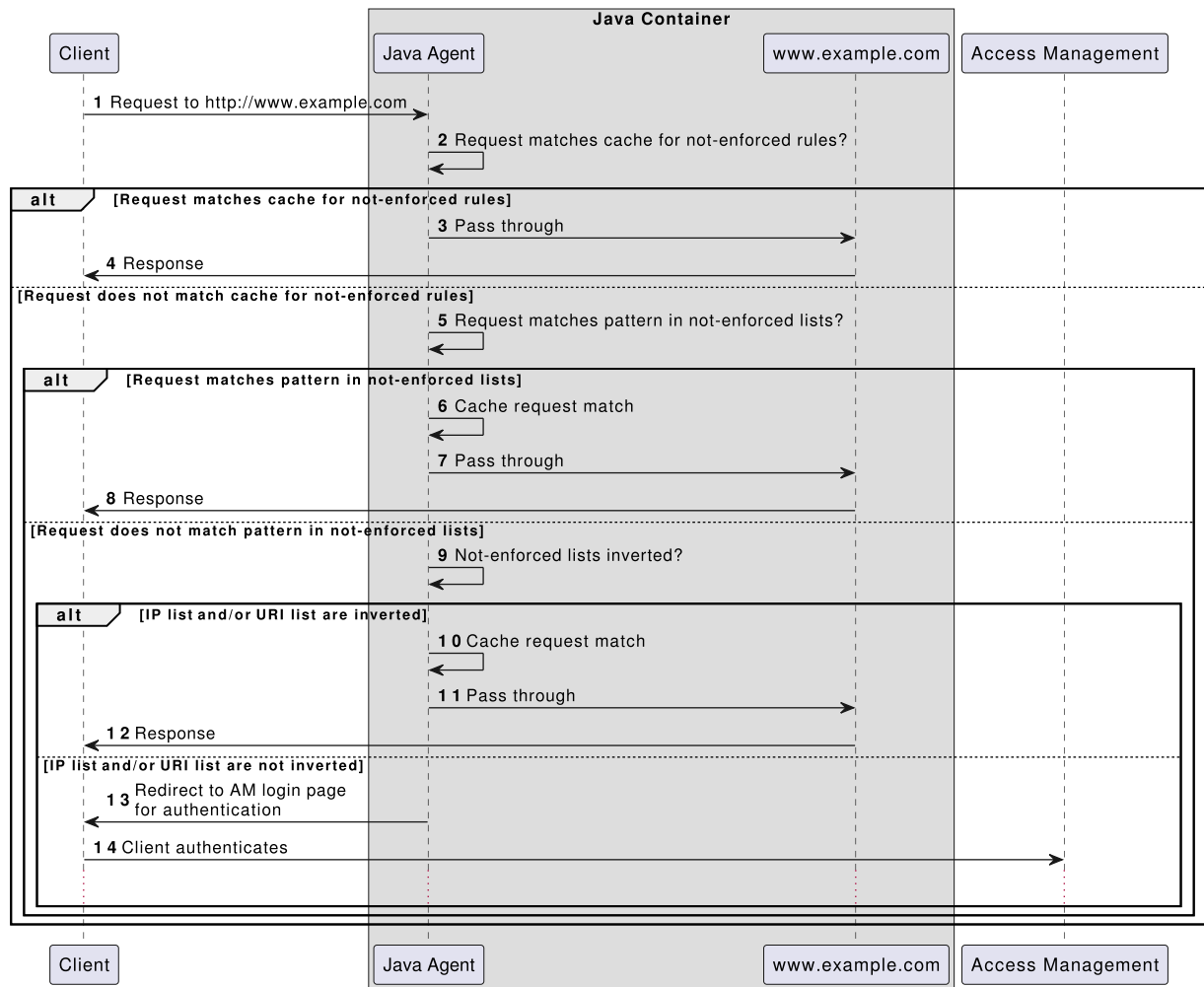
### Not Enforced Rules

It is sometimes necessary to allow unauthenticated users, or to users from specific IP addresses, to access specific resources. For example, unauthenticated users should be able to access the HTML files for the public front end of a website. Furthermore, allowing unauthenticated access to non-sensitive resources reduces the overhead of confirming each access with AM.

Java Agent manages unauthenticated access through the configuration of *not-enforced rules*. When a not-enforced rule applies to a request, the agent allows the request:

- If an unauthenticated user sent the request, the agent does not redirect the user to log in.
- If an authenticated user sent the request, the agent does not request a policy evaluation from AM.

The following image shows the data flow when Java Agent evaluates not-enforced rules for a request, first searching for a match in the cache, then in the not enforced lists:



1. A client requests a resource.

2-4. If the not-enforced URI or IP cache is enabled, the Java Agent checks whether the request matches any cached results. If the same request from the client previously matched a not-enforced rule, the Java Agent passes the request without requiring the client to authenticate.

5. If the caches are not enabled, or the request doesn't match a cached result, the Java Agent checks whether the request matches a rule in a not-enforced list.

The Java Agent evaluates every rule in the lists in order, until it finds the first match. When it finds a match, it stops evaluating, and does not consider other rules further down the list even if they provide a better match. Take care to place your most specific rules at or near the beginning of the list.

6-8. The Java Agent caches the result and passes the request without requiring the client to authenticate.

9-14. If the request doesn't match a rule in a not-enforced list, the Java Agent checks whether rules are inverted, and responds as follows:

Not-enforced URI rules	Not-enforced IP rules	Pass request without requiring authentication?
Inverted	Inverted	Yes
Not inverted	Not inverted	No
Inverted	Not inverted	No
Not inverted	Inverted	No

## *Configure Not-Enforced Rules*

Configure not-enforced rules by using the agent properties listed in [Not-Enforced Rules](#) in the *Properties Reference*, or on the Application tab of the AM console. Configure the following lists of not-enforced rules:

### ***Not-enforced URI rules***

Allow access to resources, such as images, stylesheets, or the HTML pages that provide the public front end of your site.

### ***Not-enforced IP rules***

Allow access to your site from an administrative IP address, an internal network range, or a search engine.

### ***Compound not-enforced URI and IP rules***

Allow access based on a combination of resources and IPs. When there are multiple lists of rules, the agent evaluates them in this order:

1. Compound rules in not-enforced URI and not-enforced IP lists
2. Rules in not-enforced IP lists
3. Rules in not-enforced URI lists

## *Conventions for Not-Enforced Rules*

Use the conventions in this section to define not-enforced URI rules and not-enforced IP rules:

### *Inverting rules*

Invert specific rules

Invert any rule in the not-enforced list by preceding it with the keyword `NOT`, separated by a space (blank) character.

In the following example, requests for a `.jpg` file in the `/private` URI require authentication:

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

In the following example, the agent defers to AM any request from the network specified by the 192.168.1.0/24 CIDR notation:

```
NOT 192.168.1.0/24
```

Invert all rules

Invert all rules in a not-enforced URI list by setting [Invert Not-Enforced IPs](#) to `true`.

## Wildcards

Wildcard for not-enforced IP rules

Add `*` to match all characters in a rule, except the question mark `?` character. The wildcard cannot be escaped.

```
192.168.*
```

For more information about using wildcards, see [Specifying Resource Patterns with Wildcards](#).

Wildcards for not-enforced URI rules

- Add `*` to match all characters in a rule, except the question mark `?` character. The wildcard cannot be escaped.

The wildcard spans multiple levels. For example:

```
/images/*  
/*.jsp?locale=*
```

Multiple forward slashes do not match a single forward slash. Therefore `*` matches `mult/iple/dirs`, but `mult/*/dirs` does not match `mult/dirs`.

- Add `-*-` to match all characters in a 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/-*-
```

- `*` and `-*-` wildcards cannot be used in the same rule, but can be used 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 evaluates 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*

Add the keyword `REGEX` followed by a blank (space) character before the URI or IP address. For example:

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

```
REGEX 192\.168\.10\.\d+
```

Consider the following points when using regular expressions:

- Wildcards and regular expressions cannot be used in the same rule.
- Using 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 is logged.

### *HTTP Methods*

Add one or more of the following keywords to the not-enforced rule to apply it when the incoming request uses a specific HTTP method: 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. When one or more HTTP methods are specified, only those methods are not-enforced; methods that are not specified are enforced.

The following example does not require authentication for any request method to 192.168.10.\* :

```
192.168.10.*
```

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 item to match.

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

```
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
```

To invert a method, precede it with an exclamation point ! character. The following examples require authentication for POST requests, but not for other HTTPS methods:

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

```
!POST 192.168.1.0/24
```

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

### *Cookie values*

Use the following syntax to apply not-enforced rules when the incoming request has a named cookie with a specified value:

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

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

- Name: Cookie name
- Value: Cookie value to search for.
- Modifiers: One or more modifiers to change the lookup method:
  - **c**: (For not-enforced URI rules only) Perform a case-insensitive search for the cookie name. By default, the search is case-sensitive.
  - **i**: Perform a case-insensitive search for the cookie value. By default, the search is case-sensitive.
  - **r**: Treat the string 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/*
```

The following example does not require authentication for requests to `192.168.*` when a cookie present on the request has the name `login_result` and the case-insensitive value `VALID`:

```
COOKIE(login_result/VALID/i) 192.168.*
```

Combine cookie filters with other filters, such as HTTP methods. 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.

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
```

The following example does not require authentication for GET, POST, and PUT HTTP requests from the client IP range 192.168.\*, when there is a cookie named internal, with a value that matches the regular expression .\*ID:

```
GET,POST,COOKIE(internal/.*ID/r),PUT 192.168.*
```

### Header values

Use the following syntax to apply not-enforced rules when the incoming request has a named header with a specified value:

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

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

- Name: Header name.
- Value: Header value to search for.
- 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 does not require authentication for access to .txt files in /yearly/2021/ when the request contains a header named ID (case-insensitive), with the value validated (case-insensitive):

```
HEADER(ID/validated/i) /yearly/2021/*.txt
```

The following example does not require authentication for access to the IP range 192.168.\* when the request includes a header named ID with the case-insensitive value validated:

```
HEADER(ID/validated/i) 192.168.*
```

Combine cookie filters with other filters, such as HTTP methods. 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.

The following example does not require authentication for GET , POST , and PUT 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):

The following example does not require authentication for GET , POST , and PUT requests from the IP address range 192.168.\* , 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
```

### Compound rules

Configure compound not-enforced rules to combine not-enforced URI and IP rules in a single rule.

Configure rules in either [Not-Enforced Client IP List](#) or [Not-Enforced URIs](#), using an IP rule or list of IP rules, a delimiter, and an URI rule or list of URI rules.

In the following example, the agent does not enforce HTTP requests from the IP range 192.168.1.1-192.168.4.3 to any file in the /images URI:

```
192.168.1.1-192.168.4.3 | /images/*
```

Consider the following points for compound rules:

- Place keywords, such as HTTP methods, NOT , and REGEX , at the beginning of the compound rule. Keywords affect both the IP and the URI rules.

In the following example, the agent does not enforce GET or POST HTTP requests from the IP range 192.168.1.1-192.168.4.3 , to any file ( \* ) in the /images URI.

```
GET,POST 192.168.1.1-192.168.4.3 | /images/*
```

In the following example, the agent defers to AM any request to a method except POST , from any IP address in the 192.168.1 subnet, to any file in the /private URI.

```
NOT,!POST 192.168.1.* | /private/*
```

- Check that both sides of a rule using the REGEX keyword can be parsed as a regular expression.

In the following example, the delimiter is `&&`, because the `|` character can lead to invalid regular expressions:

```
POST, REGEX 192\.168\.10\.(10|\d) && \/images\/([\^\/])+\.*\.
```

For information about configuring a different delimiter, see [Not-Enforced Compound Rule Separator](#).

- The agent caches hits and misses for each resource accessed.

Caching is enabled if either [Enable Not-Enforced IP Cache](#) or [Enable Not-Enforced URIs Cache](#) is `true`.

The cache size takes the biggest value of [Max Entries in Not-Enforced IP Cache](#) or [Max Entries in Not-Enforced URI Cache](#).

### *Extended characters*

URLs defined in [Not-Enforced URIs](#) can contain any number of extended ASCII characters. The agent container automatically percent-encodes extended characters, before the agent is called.

Extended characters in the resource path of a not-enforced rule

By default, Java Agent uses UTF-8 to percent-encode extended characters in the resource paths of not-enforced rules. To change the character encoding, set [Container Character Encoding](#).

In the following example, the agent does not enforce authentication or request policy evaluation for HTTP requests to the URL `http://www.example.com/forstå`:

```
org.forgerock.agents.notenforced.uri.list=http://www.example.com/forstå/*
```

Note how the extended ASCII character `å` can be entered without encoding.

Extended characters in HTTP query parameters of a not-enforced rule

By default, Java Agent uses ISO-8859-1 to encode extended characters in HTTP query parameters of not-enforced rules. To change the character encoding, set [Container Parameter Encoding](#).

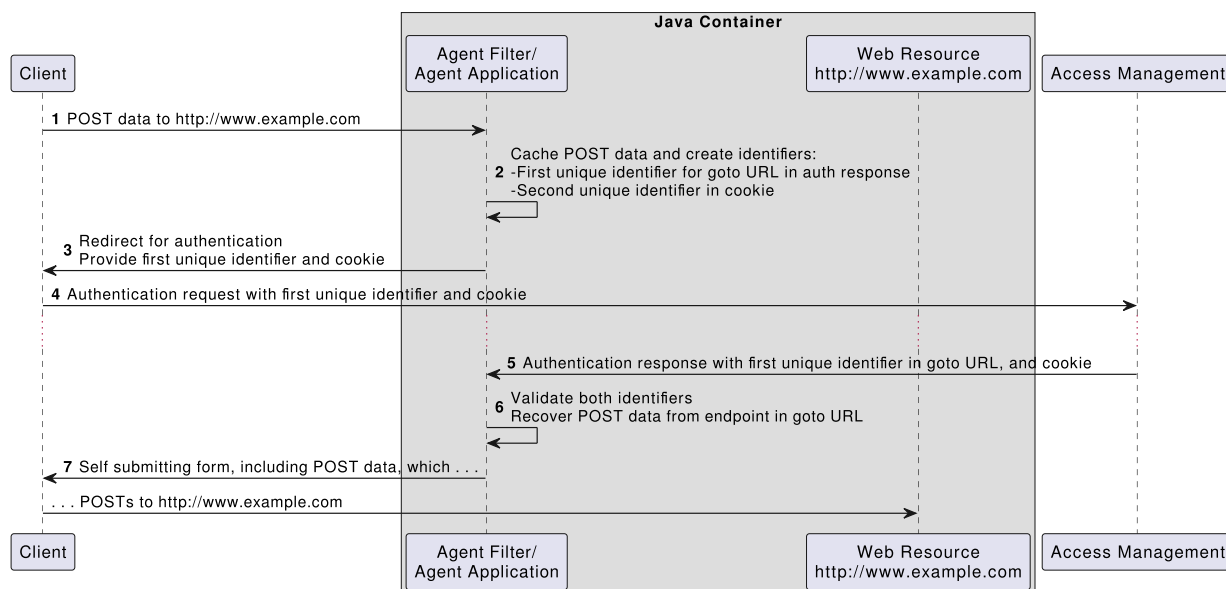
## POST Data Preservation

When POST data preservation is enabled, and an unauthenticated client posts HTML form data to a protected resource, the agent stores the data in its cache, and redirects the client

to the login screen. After successful authentication, the agent recovers the data stored in the cache, and automatically submits it to the protected resource.

Use POST data preservation in environments where clients submit form data, and have short-lived sessions.

The following image shows a simplified data flow, when an unauthenticated client POSTs data to a protected web application:



Java Agent guarantees the integrity of the data, and the authenticity of the client as follows:

1. An unauthenticated client requests a POST to a protected resource.
2. The agent caches the POST data, and generates the following unique identifiers:
  - o An identifier in the goto URL for the authentication response
  - o An identifier in a cookie

The use of two unique identifiers provides robust security, because a hacker must steal the goto URL and the cookie.

3. The agent redirects the client to AM for authentication, and includes the cookie in the redirect.
4. The client authenticates with AM.
5. AM provides an authentication response to the goto URL with the unique identifier, and includes the cookie.
6. The agent validates both identifiers, and recovers the POST data from the dummy internal endpoint given in the goto URL.

If the goto URL contains the incorrect identifier, or cannot provide a cookie containing the correct second identifier (for example, because it has expired), the agent denies the request.

The presence of the unique identifier in the goto URL ensures that requests at the URL can be individually identified. Additionally, the code makes it more difficult to hijack user data, because there is little chance of guessing the code within the login window.

7. The agent sends a self-submitting form to the client browser, that includes the form data the user attempted to post in step 1. The self-submitting form POSTs to the protected resource.

### *Configure POST Data Preservation*

Configure POST data preservation by using the agent properties listed in [POST Data Preservation](#) in the *Properties Reference*, or on the Advanced tab of the AM console.

### *Defend Against CSRF Attacks When Using POST Data Preservation*

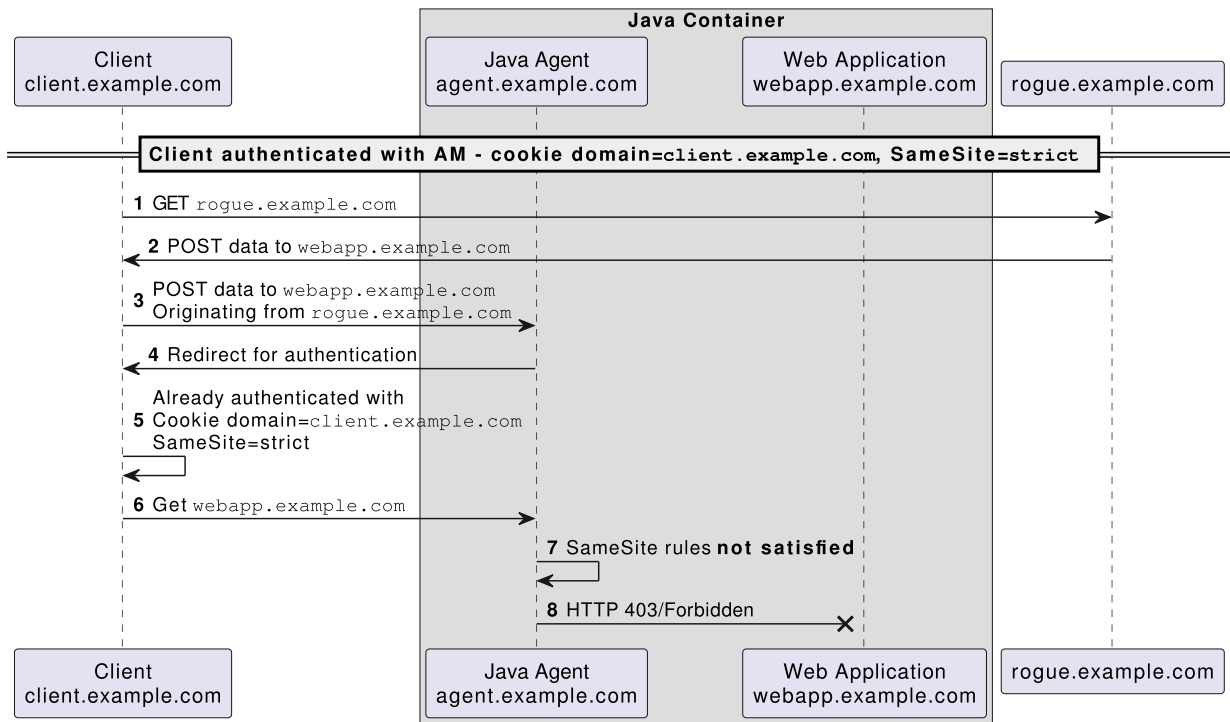
#### WARNING

Cross-site request forgery attacks (CSRF or XSRF) can be a cause of serious vulnerabilities in web applications. It is the responsibility of the protected application to implement countermeasures against such attacks, because Java Agent cannot provide generic protection against CSRF. ForgeRock recommends following the latest guidance from the [OWASP CSRF Prevention Cheat Sheet](#).

When POST data preservation is enabled, captured POST data that is replayed appears to come from the same origin as the protected application, not from the site that originated the request. Therefore, CSRF defenses that rely solely on checking the origin of requests, such as SameSite cookies or Origin headers, are not reliable. ForgeRock strongly recommend using token-based mitigations against CSRF, and relying on other measures only as a defense in depth, in accordance with OWASP guidance.

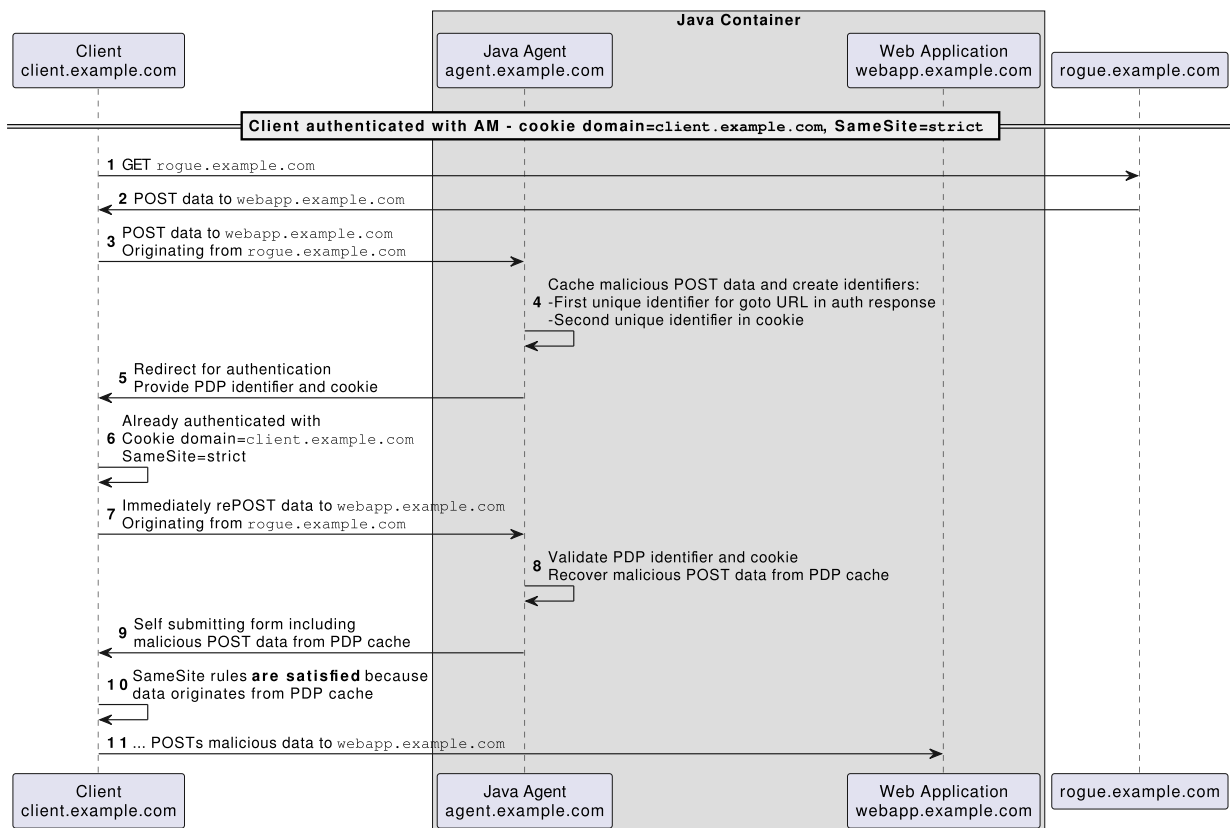
### *CSRF Attack When POST Data Preservation Is Disabled*

The following image shows a simplified data flow during a CSRF attack on an authenticated client when POST data preservation is disabled. In this limited scenario, the agent SameSite setting is enough to defend the web application:



### CSRF Attack When POST Data Preservation Is Enabled

The following image shows a simplified data flow during a CSRF attack on an authenticated client when POST data preservation is enabled. In this scenario, the SameSite setting is **not** enough to defend the web application:



## Continuous Security



When a user requests a resource through AM, excluding proxies and load balancers, the Java Agent is usually the first point of contact. Because Java Agent is closer to the user than AM, and outside the firewalls that separate the user and AM, the Java Agent can sometimes gather information about the request, which AM cannot access.

When the Java Agent requests a policy decision from AM, it can include this information in an *environment map*, a set of name/value pairs that describe the request IP and DNS name, along with other, optional, information.

In the Java Agent, use continuous security to configure an environment map. In AM, use server-side authorization scripts to access the environment map, and write scripted conditions based on cookies and headers in the request.

For information about server-side authorization scripts, see [Scripting a Policy Condition](#) in AM's *Authorization Guide*.

### *Environment Maps With Customizable Keys*

In the Java Agent, use the continuous security properties [Client Hostname Header](#) and [Client IP Address Header](#) to configure an environment map with custom keys.

The environment map has the following parts:

#### ***requestIp***

The IP address of the inbound request, determined as follows:

- If [Client IP Address Header](#) is configured, the Java Agent extracts the IP address from the header.
- Otherwise, it uses the Java function `HttpServletRequest.getRemoteAddr` to determine the IP address.

This entry is always created in the map.

#### ***requestDNSName***

The host name address of the inbound request, determined as follows:

- If [Client Hostname Header](#) is configured, the Java Agent extracts the host name from the header.
- Otherwise, it uses the Java function `HttpServletRequest.getRemoteHost` to determine the host name address.

This entry is always created in the map.

#### ***Other variable names***

An array of cookie or header values, configured by the continuous security properties [Client Hostname Header](#) and [Client IP Address Header](#).

An entry is created for each value specified in the continuous security properties.

In the following example, the continuous security properties are configured to map values for the `ssid` cookie and `User-Agent` header to fields in an environment map:

```
org.forgerock.agents.continuous.security.cookies.map[ssid]=mySSID
org.forgerock.agents.continuous.security.headers.map[User-Agent]=myUser-Agent
```

If the incoming request contains an `ssid` cookie and a `User-Agent` header, the environment map takes the value of the cookie and header, as shown in this example:

```
requestIp=192.16.8.0.1
requestDnsName=client.example.com
mySSID=77xe99f4zqi1199z
myUser-Agent=Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0)
like Gecko
```

### *Environment Maps With Fixed Keys*

In the Java Agent, use the continuous security properties [GET Parameter List for URL Policy Env](#), [POST Parameter List for URL Policy Env](#), and [JSession Parameter List for URL Policy Env](#) to configure an environment map with fixed keys.

## Attribute Fetch Modes

For information about properties to configure attribute fetching, see the [Properties Reference](#).

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

You can configure the type of attributes to fetch, and map the attribute names used on AM to the values used in the containers. The agent securely fetches the user and session data from the authenticated user, as well as policy response attributes.

## Autonomous Mode

In autonomous mode, the agent operates independently of AM, without needing to contact an AM instance. Agents allow access to resources as defined in not-enforced lists; otherwise, they deny access.

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 is not available in autonomous mode:

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

To enable autonomous mode, in the bootstrap properties file, `AgentBootstrap.properties`, set `Autonomous mode` to `true`, and restart the Java container where the agent is installed.

#### IMPORTANT

Because the agent does not contact AM when it starts in autonomous mode, the value of `Location of Agent Configuration Repository` must be `LOCAL`.

## Login Attempt Limits

For information about properties to configure attempt limits, see the [Properties Reference](#).

When a client does not present a valid SSO token, the Java Agent redirects the user to the login URL configured in AM. To mitigate redirect loops that can result in an error page presented to the end-user, configure the to limit the number login attempts.

Configure the following properties to limit login attempts and redirects:

- `Login Attempt Limit`, to specify a non-zero value for the number of login attempts. For example, if the property is set to 3, then the agent blocks the fourth login request.
- `Redirect Attempt Limit`, to limit the number of redirections the agent can take for a single browser session.

## FQDN Checking

For information about properties to configure FQDN checking, see the [Properties Reference](#).

Clients must use valid URLs with fully qualified domain names (FQDNs) to access resources. If a client uses an invalid URL, policy evaluation can fail because the FQDN does not match the requested URL, and the request fails. Misconfigured URLs can also cause incorrect policy evaluation for subsequent access requests.

In load-balanced and virtual host environments, clients can specify resource URLs that differ from the FQDNs stored in AM policies. The Java Agent provides the following properties:

- [Default FQDN](#): Specifies a default URL with valid hostname. If the agent finds an invalid URL in the client request, it redirects the request to this URL.
- [FQDN Map](#): Maps invalid URLs, load balanced URLs, and virtual host URLs to valid URLs. Each entry in the map has precedence over the FQDN default.

To prevent redirects to the FQDN default, map all anticipated invalid URLs.

## Cookie Reset Properties

For information about properties to configure cookie reset, see the [Properties Reference](#).

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 agent and another authentication system. The agent can reset cookies set by the other mechanism before redirecting the client to a login page.

The cookie reset properties include the following:

- List of cookies that can be reset
- Domain map specifying the domains for each cookie
- Path map specifying the path from which the cookie is 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.

## Caching

Java Agent allocates memory from the Java heap space in the web container to the following caches:

### *Configuration Cache*

When the agent starts up in remote configuration mode, it retrieves a copy of the agent profile from AM, and stores it in the cache. The cached information is valid until one of the following events occurs:

- AM notifies the agent of changes to hot-swappable agent configuration properties. The agent flushes the configuration cache and rereads the agent profile from AM.
- The agent restarts.
- The agent rereads the configuration from AM or from local files at the frequency specified by Configuration Reload Interval.

If the reload interval is disabled, and notifications are disabled, the cached configuration remains valid until the 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 occur:

- 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 Session Cache TTL.

### *Policy Decision Cache*

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

- If the client session is valid, the agent requests a policy decision from AM and then enforces it.
- If the client session is not valid, the 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.

After the client authenticates, the agent requests policy decision from AM and enforces it.

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

### *Session and Policy Validity in Cache*

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

Event	What is invalidated?
Client logs out from AM (and session notifications are enabled)	Session and policy decisions related to the session
Policy decision reaches the expiration time specified by <a href="#">Policy Cache TTL</a>	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 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 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 Agent maintains a cache of hit and miss for each of the not-enforced lists specified in [Not-Enforced Rules](#).

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 rule evaluations 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.

### *POST Data Preservation Cache*

When POST data preservation is enabled, the 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 in [POST Data Preservation Cache TTL](#).

For more information, see [POST Data Preservation](#).

### *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, agents cache decoded JWTs.

When an 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 agent retrieves the decoded JWT from the cache and continues processing the request.
- The hash is not in the cache. The 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 [JWT Cache TTL](#).

## Query Parameter Handling

For information about properties to configure query parameter handling, see the [Properties Reference](#).

By default, Java Agent considers any query parameters to be part of the URL, and inserts the entire string into the policy decision cache. For example, the agent inserts 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
```

Web applications that add new parameters to the URL on every request would fill the 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, configure agents to do one of the following:

- Retain nominated URL parameters. For example, remove all except that are added as part of the policy evaluation.

Configure one of the following properties:

- [Query Parameter List for Policy Evaluation](#)
- [Regex Query Parameters List for Policy Evaluation](#)
- Discard nominated URL parameters. For example, to remove all parameters added by the `angular.js` framework.

Configure one of the following properties:

- [Remove Query Parameters List for Policy Evaluation](#)
- [Regex Remove Query Parameters List for Policy Evaluation](#)

The properties are mutually exclusive, and the agent checks 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 Agent strips 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.

## Authentication Failure

For information about properties to configure notifications for authentication failure, see [Authentication Failure](#).

To limit the amount of information available to malicious users, by default Java Agent returns an HTTP 400 message when it or AM cannot authenticate the end user, regardless of the reason.

If, for example the agent returns an "unknown user" message, malicious users can use that information to try with different user names until the error message changes to, for example, "wrong password".

The following table summarizes reasons for the agent to return an HTTP 400 message:

Reason Code	Meaning
AUTHN_BOOKKEEPING_COOKIE_MISSING	The agent cannot find the authentication tracking cookie, defined in <a href="#">Pre-Authentication Cookie Name</a> . This error can happen if the user successfully authenticates, but clicks the back button of the browser to return to the previous page.



Reason Code	Meaning
NONCE_MISSING	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.
BAD_AUDIENCE	The audience in the JWT does 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	The agent found the session ID token, but threw an exception while parsing it. Alternatively, the agent cannot connect to AM to validate the ID token, maybe due to a network outage.

An HTTP 400 message is not always helpful for debugging the agent flow or when another web application depends on the error message. To customize agent behaviour, 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 agent to return an HTTP 400 message.

## Connection Pooling

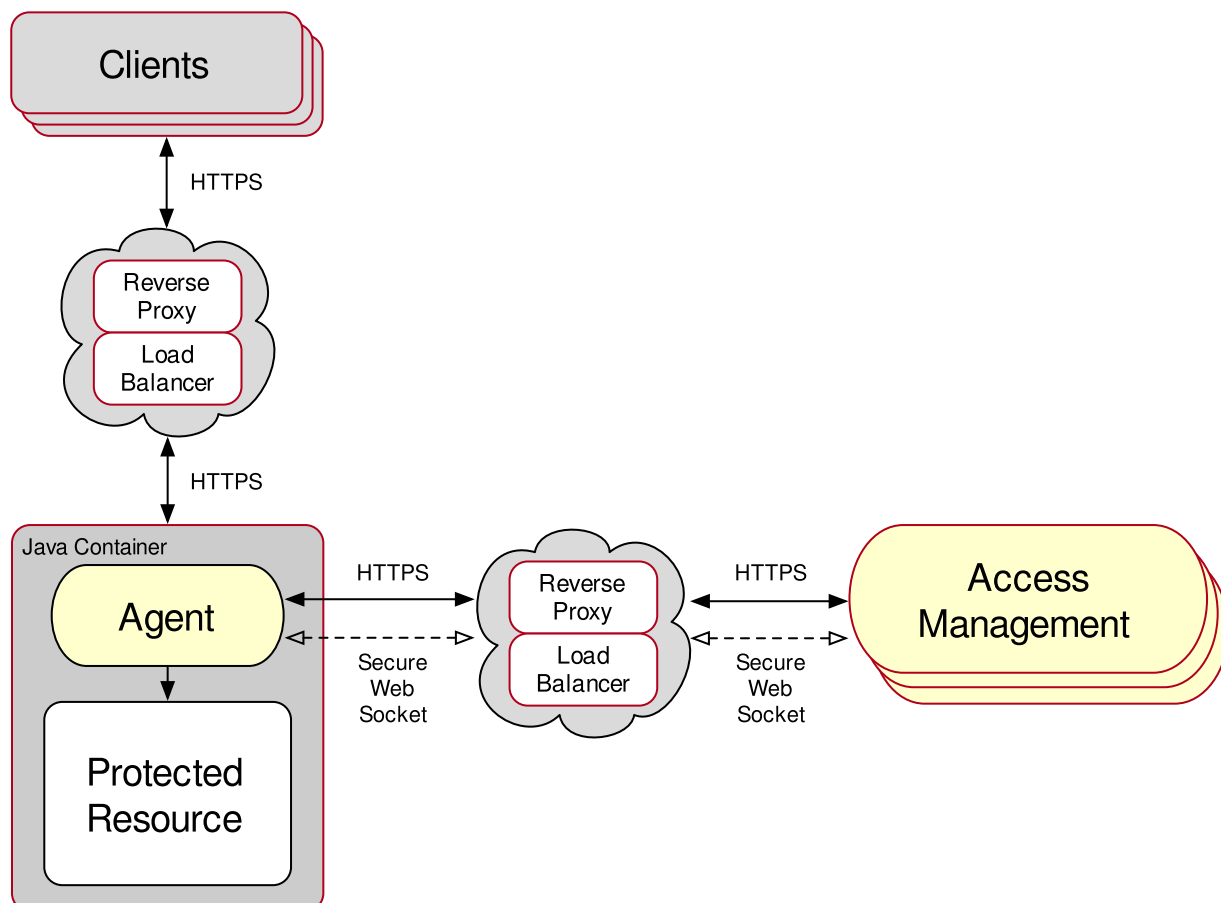
For information about properties to configure notifications for authentication failure, see the [Properties Reference](#).

Use connection pooling to control the number of connections made by the Java Agent to AM. When AM is available over a high-bandwidth connection, connection pooling could

possibly reduce performance.

## Configuration for Load Balancers and Reverse Proxies

Most environments deploy a load balancer and reverse proxy between the agent and clients, and another between the agent and AM, as shown in the following diagram:



The reverse proxy and the load balancer can be the same entity. In complex environments, multiple load balancers and reverse proxies can be deployed in the network.

## Identifying Clients Behind Load Balancers and Reverse Proxies

When a load balancer or reverse proxy is situated in the request path between the agent and a client, the agent does not have direct access to the IP address or hostname of the client. The agent cannot identify the client.

For load balancers and reverse proxies that support provision of the client IP and hostname in HTTP headers, configure the following properties:

- [Client IP Address Header](#)
- [Client Hostname Header](#)

When there are multiple load balancers or reverse proxies in the request path, the header values can include a comma-separated list of values, where the first value represents the client, as in `client,next-proxy,first-proxy`.

## Agent - Load Balancer/Reverse Proxy - AM

When a reverse proxy is situated between the agent and AM, it protects the AM APIs.

When a load balancer is situated between the agent and AM, it regulates the load between different instances of AM.

Consider the points in this section when installing Java Agent in an environment where AM is behind a load balancer or a reverse proxy.

### *Agent's IP Address and/or FQDN*

The load balancer or reverse proxy conceals the IP addresses and FQDNs of the agent and of AM. Consequently, AM cannot determine the agent base URL.

Do the following to prevent problems during installation, or with redirection using the `goto` parameter:

- Configure the load balancer or reverse proxy to forward the agent IP address and/or FQDN in a header.
- Configure AM to recover the forwarded headers. For more information, see [Configuring AM to Use Forwarded Headers](#).
- Install the agent using the IP address or FQDN of the load balancer or reverse proxy as the point of contact for the AM site.

### *AM Sessions and Session Stickiness*

Improve the performance of policy evaluation by setting AM's sticky cookie (by default, `am1bcookie`) to the AM's server ID. For more information, see [Configuring Site Sticky Load Balancing in AM's Setup Guide](#).

When configuring multiple agents, consider the impact on sticky load balancer requirements of using one or multiple agent profiles:

- If the agents are configured with multiple agent profiles, configure sticky load balancing. This is because the agent profile name is contained in the OpenID Connect JWT, used by the agent and AM for communication. Without session stickiness, there is no way to make sure that the appropriate JWT ends in the appropriate agent instance.
- If multiple agents are configured with the same agent profile, decide whether to configure sticky load balancing depending on other requirements of your

environment.

## WebSockets

For communication between the agents and the AM servers, the load balancers and reverse proxies must support the WebSocket protocol. For more information, see the load balancer or proxy documentation.

### TIP

For an example of how to configure Apache HTTP as a reverse proxy, see [Configure Apache Server As a Reverse Proxy Example](#).

## Configure AM to Use Forwarded Headers

When a load balancer or reverse proxy is situated between the agent and AM, configure AM to recover the forwarded headers that expose the agents' real IP address or FQDN.

1. Log in to the AM console as an administrative user, such as `amAdmin`.
2. Select Realms > **Realm Name** > Services.
3. Select Add a Service > Base URL Source > 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 other fields empty.

For more information about the Base URL Source service, see [Base URL Source](#) in AM's *Reference*.

5. Save your changes.

## Agent - Load Balancer/Reverse Proxy - Client

When a reverse proxy is situated between the agent and client, it renders anonymous the client traffic that enters the network.

When a load balancer is situated between the agent and client, it regulates the load between the agents and the containers.

Consider the points in this section when installing Java Agent in an environment where clients are behind a load balancer or a reverse proxy:

### *Forward Client's IP Address and/or FQDNs*

The load balancer or reverse proxy conceals the IP addresses and FQDNs of the agent and clients. Consequently, the agent cannot determine the client base URL.

Configure the load balancer or reverse proxy to forward the client IP address and/or the client FQDN in a header. Failure to do so prevents the agent from performing policy evaluation, and applying not-enforced and conditional login/logout rules.

For more information, see [Configuring Client Identification Properties](#).

### *Use Sticky Load Balancing With POST Data Preservation*

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

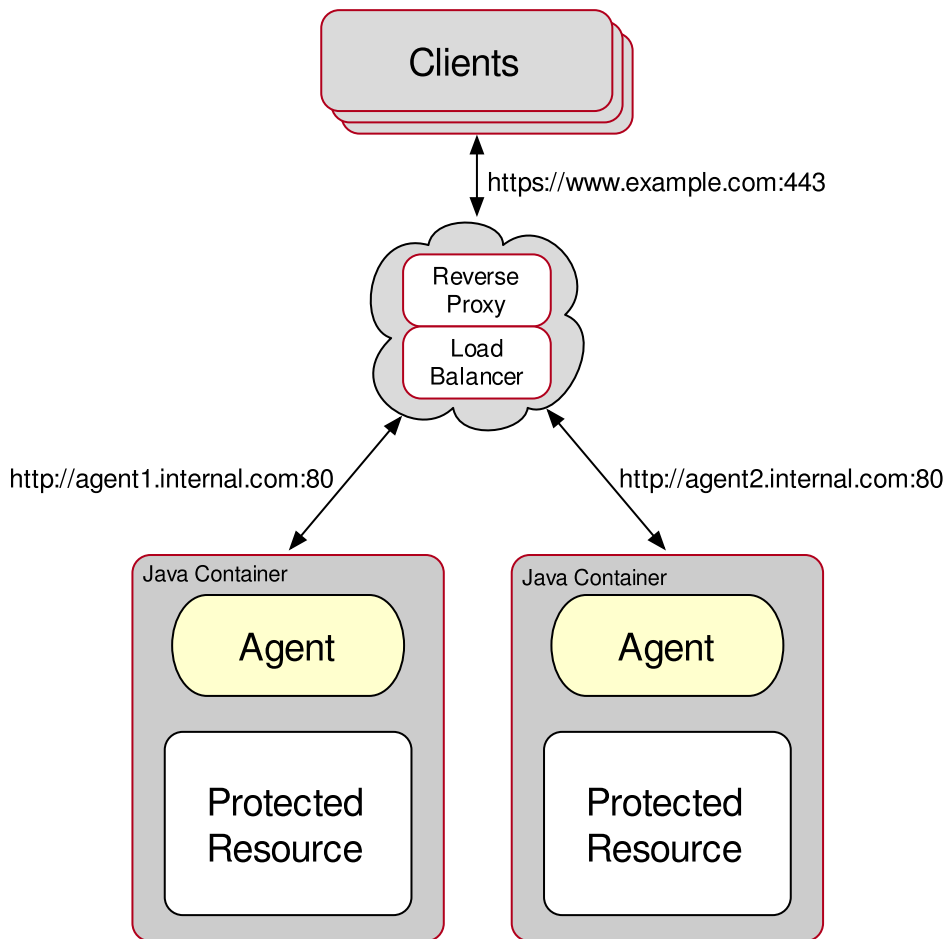
Agents provide properties to set either sticky cookie or URL query string for load balancers and reverse proxies.

For more information, see [Configuring POST Data Preservation for Load Balancers or Reverse Proxies](#).

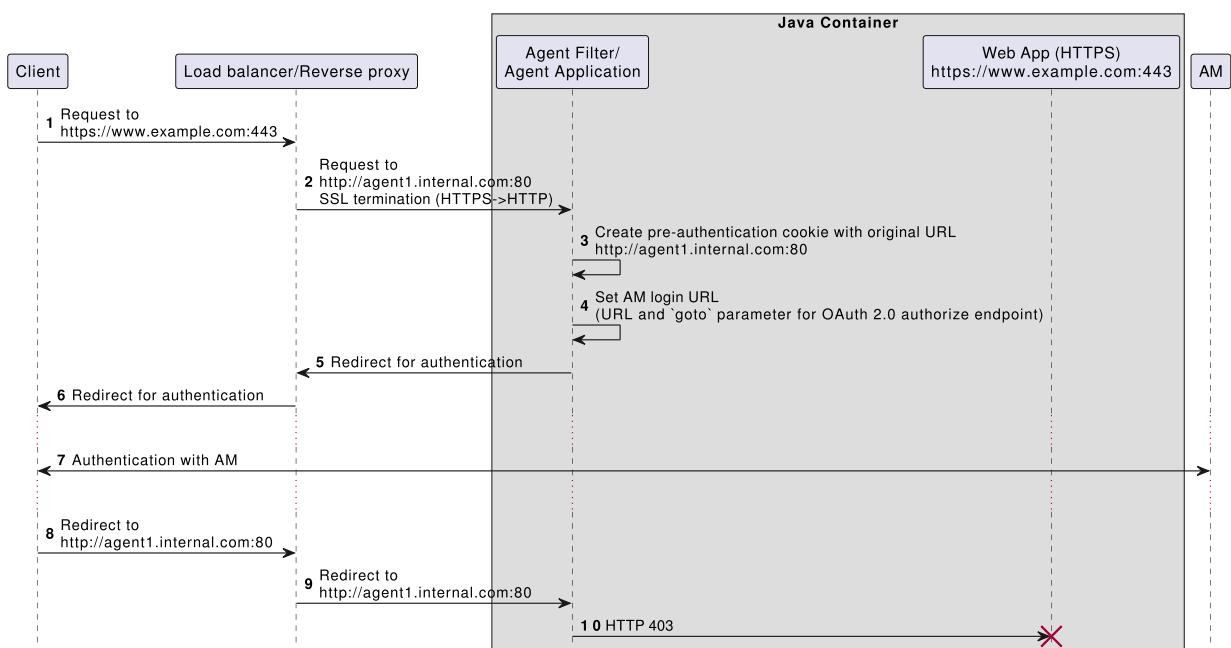
### *Override Protocol, Host, and Port After SSL Offloading*

The load balancer or reverse proxy 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, because the public key is usually processed by a hardware accelerator.

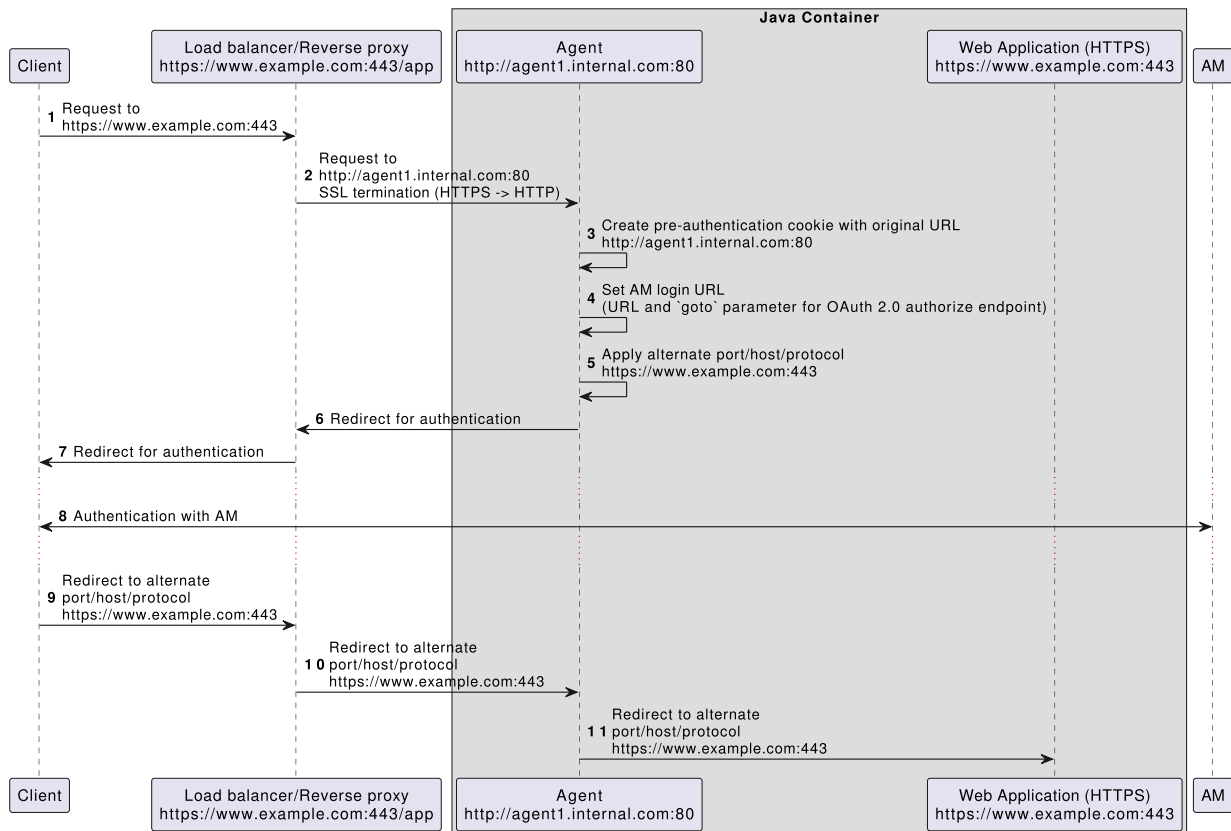
The following diagram shows the agent connected to a client through a reverse proxy and load balancer. The agent connection to the reverse proxy and load balancer is on HTTP and port 80. The client connection is on HTTPS and port 443.



After SSL offloading, the host, port, and protocol of the request is changed to match the request received by the agent; it no longer matches the request from the client, as shown in the following data flow. The agent uses this URL for the redirect\_url from the OAuth 2.0 flow, which causes the request to fail.



In the following flow, the agent overrides the host, port, and protocol for subsequent redirects:

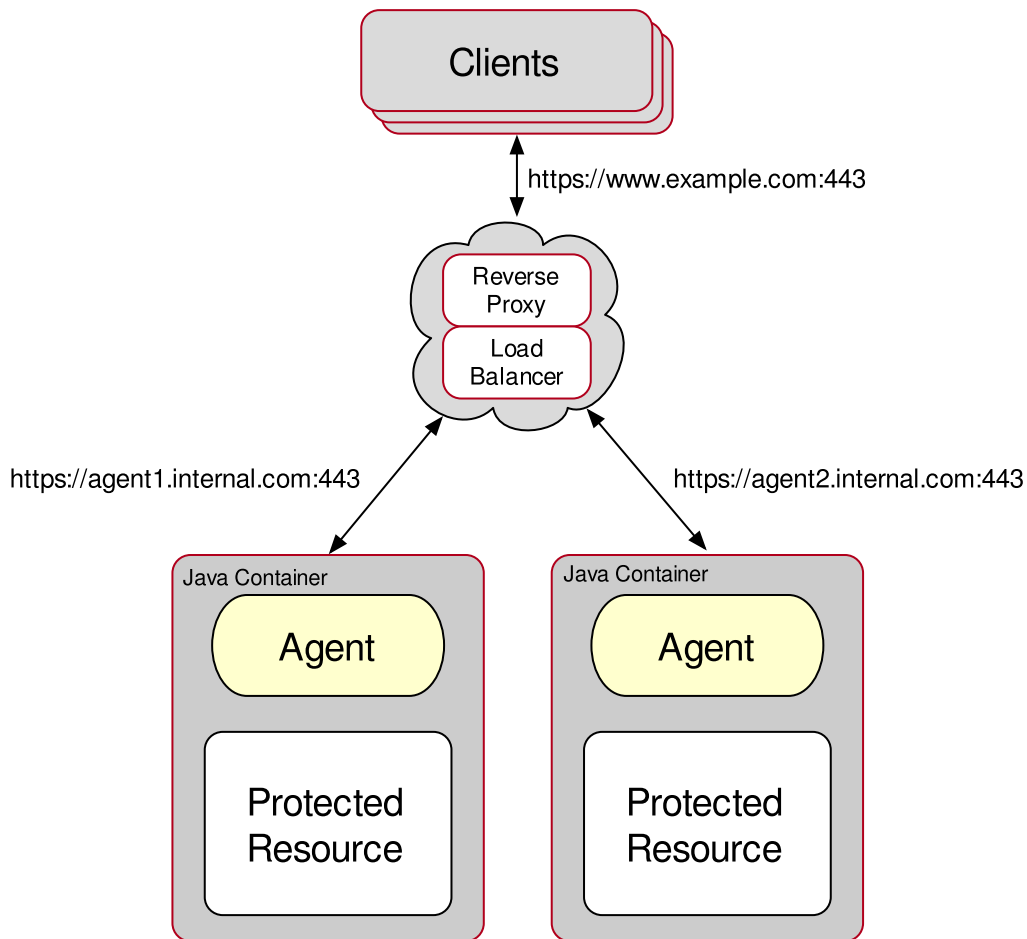


For this scenario, configure the agent as described in [To Override Protocol, Host, and Port](#).

### *Match FQDNs for Request Forwarding*

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 following diagram illustrates this scenario:



For this scenario, configure the agent as described in [To Map the Agent Host Name to the Load Balancer or Reverse Proxy Host Name](#).

### *Override Protocol, Host, and Port*

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

The agent in this example is in [remote configuration mode](#), but the steps mention properties for agents in [local configuration mode](#).

#### **IMPORTANT**

Agent configuration for SSL offloading prevents FQDN checking and mapping. Consequently, URL rewriting and redirection do not work correctly when the agent is accessed directly and not through the load balancer or proxy. This should not be a problem for client traffic, but could be a problem for web applications accessing the protected container directly, from behind the load balancer.

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



2. Select Realms > **Realm Name** > Applications > Agents > Java > **Agent Name** > Advanced.

3. Set **Alternative Agent Host Name** to that of the load balancer or reverse proxy. For example, `lb.example.com`.

The equivalent property setting is

`org.forgerock.agents.agent.hostname=lb.example.com`.

4. Set **Alternative Agent Port** to that of the load balancer or proxy. For example, `80`.

The equivalent property setting is `org.forgerock.agents.agent.port=80`.

5. Set **Alternative Agent Protocol** to that of the load balancer or proxy. For example, `http` or `https`.

The equivalent property setting is

`org.forgerock.agents.agent.protocol=https`.

6. Save your work.

7. Restart the Java container where the agent is installed.

## *Map the Agent Host Name to the Load Balancer or Reverse Proxy Host Name*

When protocols and port numbers match, configure FQDN mapping.

The agent in this example is in remote configuration mode, but the steps mention properties for agents in local configuration mode.

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

2. Select Realms > **Realm Name** > Applications > Agents > Java > **Agent Name**.

3. In the Global tab, enable FQDN Check.

The equivalent property setting is

`org.forgerock.agents.fqdn.check.enabled=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

`org.forgerock.agents.fqdn.default=lb.example.com`.

5. Append the FQDN of the load balancer or proxy to the field **Agent Root URL for CDSO**.

6. Map the load balancer or proxy FQDN to the FQDN where the 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

```
org.forgerock.agents.fqdn.map[agent.example.com]=lb.example.com .
```

7. Save your work.
8. Restart the Java container where the agent is installed.

## *Configure Client Identification Properties*

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

This procedure explains how to configure the client identification properties.

The agent in this example is in remote configuration mode, but the steps mention properties for agents in local configuration mode.

1. Log in to the AM console with a user that has permissions to modify the Java agent profile.
2. Select Realms > **Realm Name** > Applications > Agents > Java > **Agent Name** > Advanced.
3. 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

```
org.forgerock.agents.http.header.containing.ip.address=X-Forwarded-For .
```

4. 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

```
org.forgerock.agents.http.header.containing.remote.hostname=X-Forwarded-Host .
```

## 5. Save your changes.

### *Configure POST Data Preservation for Load Balancers or Reverse Proxies*

Configure the load balancer or reverse proxy **and** the agents for session stickiness.

The agent in this example is in remote configuration mode, but the steps mention properties for agents in local configuration mode.

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

In **PDP Sticky session mode**, configure one of the following options:

- **Cookie:** The agent creates a cookie for POST data preservation session stickiness. The contents of the cookie is configured in the next step.
- **URL:** The agent appends to the URL a string specified in the next step.

The equivalent property setting is

```
org.forgerock.agents.pdp.sticky.session.mode=Cookie|URL ] .
```

4. In the POST Data Preservation Sticky Session Key Value property, 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 .
```

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

## Troubleshooting

ForgeRock provides support services, professional services, training through ForgeRock University, and partner services to help you set up and maintain your deployments. For information about getting support, see [Getting support](#).

When you are trying to solve a problem, save time by asking the following questions:

- How do you reproduce the problem?
- What behavior do you expect, and what behavior do you see?
- When did the problem start occurring?
- Are their circumstances in which the problem does not occur?
- Is the problem permanent, intermittent, getting better, getting worse, or staying the same?

If you contact ForgeRock for help, include the following information with your request:

- Description of the problem, including when the problem occurs and its impact on your operation.
- The product version and build information.
- Steps you took to reproduce the problem.
- Relevant access and error logs, stack traces, and core dumps.
- Description of the environment, including the following information:
  - Machine type
  - Operating system and version
  - Web server or container and version
  - Java version
  - Patches or other software that might affect the problem

## Questions and Answers

### ▼ [Cannot Install Over HTTPS](#)

#### Question:

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 (&lt;) 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?

**Answer:**

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 [Configuring AM's Container for HTTPS](#) of *AM's Installation Guide*.

#### ▼ [Cannot Install WebSphere Java agent on Linux](#)

**Question:**

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?

**Answer:**

Edit **agentadmin** to use IBMJCE, and then try again. For information, see [Install With IBM Java](#).

## ▼ [Infinite Redirection Loops With Stateless Sessions](#)

### Question:

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:

```
<timestamp> +0000 ERROR [c53...348]state identifier not present
in authentication state
<timestamp> +0000 WARNING [c53...348]unable to verify pre-
authentication cookie
<timestamp> +0000 WARNING
[c53...348]convert_request_after_authn_post(): unable to
retrieve pre-authentication request data
<timestamp> +0000 DEBUG [c53...348] exit status: forbidden (3),
HTTP status: 403, subrequest 0
```

What is happening?

### Answer:

This redirection loop happens because the client-based (stateless) session cookie is surpassing the maximum supported browser header size. Because 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 AM's [Security Guide](#).

## ▼ [Redirection URI Error After Upgrade](#)

### Question:

I have upgraded my agent and I see the following message in the agent log:

```
redirect_uri_mismatch. The redirection URI provided does not
match a pre-registered value.
```

What should I do?

### Answer:

Java Agent accept only 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, 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 [Alternative Agent Protocol](#) to access the agent through different protocols. For example, `http://agent.example.com/` and `https://agent.example.com/`.
- Configuring [Alternative Agent Host Name](#) to access the agent through different virtual host names. For example, `http://agent.example.com/` and `http://internal.example.com/`.
- Configuring [Alternative Agent Port Number](#) to access the agent through different ports. For example, `http://agent.example.com/` and `http://agent.example.com:8080/`.

#### ▼ [File Not Found Errors After WebSphere Installation](#)

##### Question:

After installing a Java Agent on WebSphere, accessing a URL for a folder in a protected web 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?

##### Answer:

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.

For more information, see the IBM documentation, [Setting webcontainer custom properties](#).

# Reference

---

## agentadmin Command

The **agentadmin** command manages Java Agent installation. It requires a Java runtime environment.

The command supports the following options:

### ***--install***

Installs a new agent instance.

Usage: **agentadmin --install [--useResponse | --saveResponse *file-name*] [--acceptLicence]**

When the command is used without options, the installation process prompts for the following information:

- Confirmation that you have read and accepted the software license agreement.
- Information about the container installation.
- The URL to the AM instance. The Java Agent confirms that it can log in to AM by using the profile name and password provided during installation. If unsuccessful, the installation stops with an error.
- The URL to the Java Agent instance. The Java Agent confirms that it can access the host and port of the URL. If the port is busy, it prompts the user to stop the container.
- The Java Agent profile name in AM.
- The AM realm containing the Java Agent profile.
- The path to the file containing the Java Agent password.

### ***--useResponse***

Run 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***

Save all the supplied responses in a response file specified by *file-name*.

### ***--acceptLicense***

Confirms automatically that you have read and accept the software license agreement, without prompting you to agree. To view the license agreement, open `<server-root>/legal-notices/license.txt.file-name` .

### ***--forceInstall***

Installs a new Java Agent instance, without checking the AM URL or Java Agent URL.



Usage: **agentadmin --forceInstall** [--useResponse | --saveResponse *file-name*] [--acceptLicence]

When the command is used without options, the installation process prompts for the following information:

- Confirmation that you have read and accepted the software license agreement.
- Information about the container installation.
- The URL to the AM instance. The Java Agent does not confirm that it can log in to AM. Take care to provide a valid URL.
- The URL to the Java Agent instance. The Java Agent does not confirm that it can access the URL. Take care to provide a valid URL.
- The Java Agent profile name in AM.
- The AM realm containing the Java Agent profile.
- The path to the file containing the Java Agent password.

***--useResponse***

Run 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***

Save all the supplied responses in a response file specified by *file-name*.

***--acceptLicense***

Confirms automatically that you have read and accept the software license agreement, without prompting you to agree. To view the license agreement, open `<server-root>/legal-notices/license.txt.file-name` .

***--custom-install***

Installs a new agent instance, specifying advanced configuration options.

Usage: **agentadmin --custom-install** [--useResponse | --saveResponse *file-name*] [--acceptLicence]

***--useResponse***

Run 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***

Save all the supplied responses in a response file specified by *file-name*.

***--acceptLicense***

Confirms automatically that you have read and accept the software license agreement, without prompting you to agree. To view the license agreement, open `<server-root>/legal-notices/license.txt.file-name` .

***--uninstall***

Uninstalls an existing agent instance.

Usage: **agentadmin --uninstall** [--useResponse | --saveResponse *file-name*]

**--useResponse**

Run 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**

Save all the supplied responses in a response file specified by *file-name*.

**--version**

Displays the version information.

**--uninstallAll**

Uninstalls all the agent instances.

**--listAgents**

Displays details of all the configured agents.

**--agentInfo**

Displays information about the agent corresponding to the specified *agent-id*.

Usage: **agentadmin --agentInfo** *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.

## Configure Apache HTTP Server As a Reverse Proxy Example

This section provides an example of how to configure Apache as a reverse proxy between AM and the agent. You can use any reverse proxy that supports the WebSocket protocol.

Refer to the Apache documentation to configure Apache for load balancing and any other requirement for your environment.

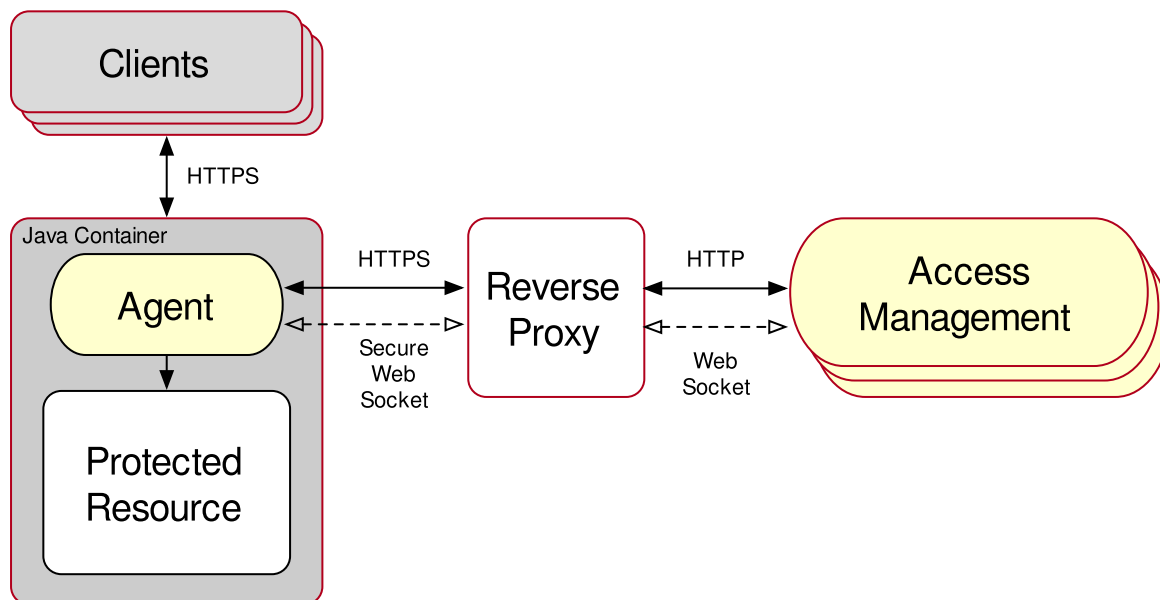


Figure 2. Reverse Proxy Configured Between the Agent and AM

Note that the communication protocol changes from HTTPS to HTTP.

### Configure Apache as a Reverse Proxy Example

1. In your deployed reverse proxy instance, locate the `httpd.conf` file.
2. Add the following modules required for a proxy configuration:

```
# 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, and set the following directives:

```
<VirtualHost 192.168.1.1>
...
# Proxy Config
RequestHeader set X-Forwarded-Proto "https" ①
ProxyPass "/openam/notifications"
"ws://openam.example.com:8080/openam/notifications"
Upgrade=websocket ②
ProxyPass "/openam"
"http://openam.example.com:8080/openam" ③
ProxyPassReverseCookieDomain "openam.internal.example.com"
```

```
"proxy.example.com" ④  
ProxyPassReverse "/openam"  
"http://openam.example.com:8080/openam" ⑤  
...  
</VirtualHost>
```

- ① **RequestHeader:** If the proxy is configured for https, set to https . Otherwise, set to http . A later step configures AM to recognize the forwarded header and use it in the goto parameter, to redirect back to the Java Agent after authentication.
  - ② **ProxyPass:** Allow WebSocket traffic between AM and the Java Agent. If HTTPS is configured between the proxy and AM, use wss instead of ws .
  - ③ **ProxyPass:** Allow HTTP traffic between AM and the agent.
  - ④ **ProxyPassReverseCookieDomain:** Rewrite the domain string of Set-Cookie headers in this format: **internal domain** (AM's domain) **public domain** (proxy's domain) .
  - ⑤ **ProxyPassReverse:** Set to the same value configured for the ProxyPass directive.
4. Restart the reverse proxy instance.
  5. Configure AM to recover the forwarded header configured in the reverse proxy. Also, review other configurations that may be required in an environment that uses reverse proxies. For more information, see [Communication Between AM and Agents](#)

## Implement a Custom Task Handler

This section describes how to add a custom task handler to the list of handlers, and provides example handlers. At startup, Java Agent tries to instantiate the specified service resolver class. If unsuccessful, it instantiates the original service resolver.

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.

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

Function	When to execute the class	What the class must implement
List<String> getPreInboundTaskHandlers()	Before all other inbound task handlers	IAmFilterTaskHandler
List<String> getPostInboundTaskHandlers()	After all other inbound task handlers	IAmFilterTaskHandler
List<String> getPreSelfRedirectHandlers()	Before all other self-redirect task handlers	IAmFilterTaskHandler
List<String> getPostSelfRedirectHandlers()	After all other self-redirect task handlers	IAmFilterTaskHandler
List<String> getPreFilterResultHandlers()	Before all other result handlers	IAmFilterResultHandler
List<String> getPostFilterResultHandlers()	After all other result handlers	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 any `isActive` function implemented by your custom handler returns `true`, if appropriate. Any handler returning `false` is evicted.

For each `InboundTaskHandler` and `SelfRedirectHandler`, the `process` function 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 a null value indicates to carry on to the other handlers.

For `FilterResultHandlers`, returning a null value causes an error.

### *Example Custom Filter Result Task Handler*

```

/*
 * Copyright 2019-2021 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 Custom Self-Redirect Task Handler

```

/*
 * Copyright 2019-2021 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.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 Custom Inbound Task Handler

```

/*
 * Copyright 2019-2021 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.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 2019-2021 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.ident
 * ity.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 Filter***

A servlet that intercepts inbound client requests to a resource, and processes them according to the value of [Agent Filter Mode Map](#).

### ***Application***

A service exposing protected resources. See [Web Application](#).

In AM policies, an application is a template that constrains the policies that govern access to protected resources. An application can have zero or more policies.

### ***Application type***

Application types act as templates for creating policy applications. Application types define the following:

- A preset list of actions and functional logic, such as policy lookup and resource comparator logic.
- 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.

### ***Autonomous Mode***

The agent operates independently of AM, without needing to contact an AM instance. Agents allow access to resources as defined in not-enforced lists; otherwise, they deny access.

### ***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.

### ***Centralized configuration mode***

Replaced by [remote configuration mode](#).

### ***Circle of trust***

Group of providers, including at least one identity provider, who have agreed to trust each other to participate in a SAML v2.0 provider federation.

### ***Client***

In OAuth 2.0, requests protected web resources on behalf of the resource owner given the owner's authorization. AM can play this role in the OAuth 2.0 authorization framework.

### ***Client-based OAuth 2.0 tokens***

After a successful OAuth 2.0 grant flow, AM returns a token to the client. This differs from CTS-based OAuth 2.0 tokens, where AM returns a *reference* to token to the client.

### ***Client-based sessions***

AM sessions for which AM returns session state to the client after each request, and require it to be passed in with the subsequent request. For browser-based clients, AM sets a cookie in the browser that contains the session information.

For browser-based clients, AM sets a cookie in the browser that contains the session state. When the browser transmits the cookie back to AM, AM decodes the session state from the cookie.

### ***Conditions***

Defined as part of policies, these determine the circumstances under which a policy applies.

Environmental conditions reflect circumstances like the client IP address, time of day, how the subject authenticated, or the authentication level achieved.

Subject conditions reflect characteristics of the subject like whether the subject authenticated, the identity of the subject, or claims in the subject's JWT.

### ***Configuration datastore***

LDAP directory service holding AM configuration data.

### ***Cross-domain single sign-on (CDSSO)***

AM capability allowing single sign-on across different DNS domains.

### ***CTS-based OAuth 2.0 tokens***

After a successful OAuth 2.0 grant flow, AM returns a *reference* to the token to the client, rather than the token itself. This differs from client-based OAuth 2.0 tokens, where AM returns the entire token to the client.

### ***CTS-based sessions***

AM sessions that reside in the Core Token Service's token store. CTS-based sessions might also be cached in memory on one or more AM servers. AM tracks these sessions in order to handle events like logout and timeout, to permit session constraints, and to notify web applications involved in SSO when a session ends.

### ***Delegation***

Granting users administrative privileges with AM.

### ***Entitlement***

Decision that defines the following:

- Which resource names can and cannot be accessed for a given identity in the context of a particular web application.
- Which actions are allowed and denied.
- 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 a web 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. The Java Agent filters requests to other applications in the container, using policies based on web application resource URLs.

### ***Local configuration mode***

The agent reads its configuration from the `AgentConfiguration.properties` file. See also [remote configuration mode](#).

The configuration mode is defined by [Location of Agent Configuration Repository](#).

### ***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***

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 policy decision point.

### ***Policy Information Point (PIP)***

Entity that provides extra information, such as user profile attributes that a policy decision point needs in order to make a decision.

### ***Principal***

Represents an entity that has been authenticated (such as a user, a device, or a web application), and is therefore distinguished from other entities.

When a [Subject](#) successfully authenticates, AM associates the Subject with the Principal.

### ***protected resource***

A resource that is not matched by a "not enforced" rule.

### ***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 web 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.

### ***Remote configuration mode***

The agent ignores the configuration in `AgentConfiguration.properties`, retains the retrieved bootstrap properties, and downloads the configuration from AM. See also [local configuration mode](#).

The configuration mode is defined by [Location of Agent Configuration Repository](#).

### ***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 web 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 to end a session once, and thereby end the session across multiple web applications.

### ***Single sign-on (SSO)***

Capability to authenticate once and gain access to multiple web applications, without authenticating again.

### ***Site***

Group of AM servers configured the same way, accessed through a load balancer layer. The load balancer handles failover to provide service-level availability.

The load balancer can also be used to protect AM services.

### ***Standard metadata***

Standard federation configuration information that you can share with other access management software.

### ***Stateless Service***

Stateless services do not store any data locally to the service. When the service requires data to perform any action, it requests it from a data store. For example, a stateless authentication service stores session state for logged-in users in a database.

This way, any server in the deployment can recover the session from the database and service requests for any user.

All AM services are stateless unless otherwise specified. See also [Client-based sessions](#) and [CTS-based sessions](#).

### ***Subject***

Entity that requests access to a resource .

When an identity successfully authenticates, AM associates the identity with the [Principal](#) that distinguishes it from other identities. An identity can be associated with multiple principals.

### ***Identity store***

Data storage service holding principals' profiles; underlying storage can be an LDAP directory service or a custom IdRepo implementation.

### ***Web Agent***

Native library installed in a web server that acts as a policy enforcement point with policies based on web page URLs.

### ***Web Application***

An application that runs on a web server, that is accessed by the user through a web browser. The web application exposes protected resources.