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:
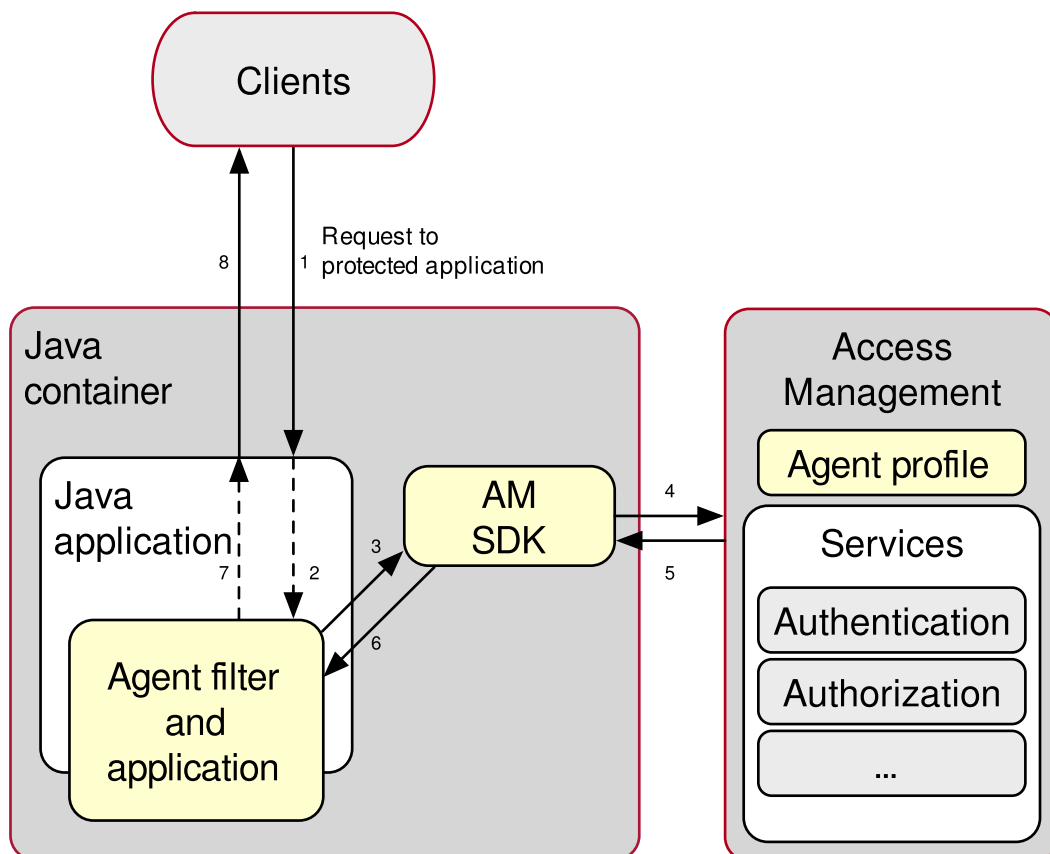
Agent configuration

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 a public and a 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 503 Service Unavailable`.

AgentConfiguration.properties

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

If the value of <u>Location of Agent Configuration Repository</u> is `LOCAL`, the agent reads the `AgentConfiguration.properties` file after `AgentBootstrap.properties`. If the value of <u>Location of Agent Configuration Repository</u> 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 <u>Encrypted Agent Password</u>.

AgentKey.properties

This file defines the following keys:

- The encryption key for the agent profile password. For more information, see <u>Encryption Key/Salt</u>.

- The signing key for pre-authentication cookies and POST data preservation cookies. For more information, see <u>Pre-Authn and Post Data Preservation Cookie Signing Value</u>.

agent-logback.xml

This file configures logging of Java Agent and third-party dependencies, using the Logback implementation of the Simple Logging Facade for Java (SLF4J) API. For more information, see <u>Logging</u>.

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 <u>Configuration Reload Interval</u>.

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 <u>Location of Agent Configuration Repository</u> 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 <u>Location of Agent Configuration Repository</u> must be `REMOTE`.

Change the agent configuration on the AM admin UI

Go to **REALMS** > realm name > **Applications** > **Agents** > **Java** > agent name.

The value of <u>Location of Agent Configuration Repository</u> must be `REMOTE`.

Realms

Agent profile realm

The agent profile stores a set of configuration properties that define the behavior of the agent.

During agent installation, the installer prompts for the profile realm, and populates the property <u>Agent Profile Realm</u> in the bootstrap properties file. By default, the profile realm is set to the top-level realm.

The agent profile realm can be different to the user and policy evaluation realms. Groups of agents can use the same agent profile realm, which can be separate from the user and policy evaluation realms.

For information about creating agent profiles in the top-level realm or other realms, see <u>Create agent profiles</u>.

Policy evaluation realm

The policy evaluation realm is the realm the agent uses to request policy decisions from AM. In most circumstances, the policy evaluation realm is the same as the user realm.

The policy evaluation realm is configured by <u>Policy Evaluation Realm Map</u>, and defaults to the top-level realm. The policy set to use is configured by <u>Policy Set Map</u> To ensure that policies are always evaluated in the user realm, set <u>Enable Policy Evaluation in User Authentication Realm</u> to `true`.

In AM, only the top-level realm has a default policy set, called iPlanetAMWebAgentService. If you use a policy evaluation realm that is in a subrealm of the top-level realm, you must also define a policy set and policies in the equivalent realm in AM.

User realm

The user realm is the realm in which a user is authenticated. In most circumstances, the user evaluation realm is the same as the policy realm.

By default, users authenticate to AM in the top-level realm, however, the agent can authenticate users in different realms depending on the request domain, path, or resource.

When a user logs out, the agent maintains the user realm. The agent obtains the realm info from the JWT, if one is available, or by calling `sessioninfo`. When the user logs out, the stored realm is passed to the logout endpoint automatically.

The first time an authenticated user requests a resource from the agent, the agent establishes the user realm from the session. It permanently associates the realm with the session in the session cache. When the session ends, the agent automatically passes the realm to the logout endpoint.

For more information about changing the user realm, see Login redirect.

Sessions

On startup, Java Agent uses the following properties to obtain a session from AM:

- Agent Profile Name
- Encrypted Agent Password
- Agent Profile Realm

The session lifetime is defined by the AM version and configuration, and is essentially indefinite.

For the security of your deployment, set the session lifetime as described in Manage sessions.

If you clear agent sessions in the AM admin UI, you can accidentally kill an active agent session. If this happens, the agent detects that its session has expired and automatically obtains a new one.

Request flow

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 more information, see Single sign-on in AM's *Authentication and SSO guide*.
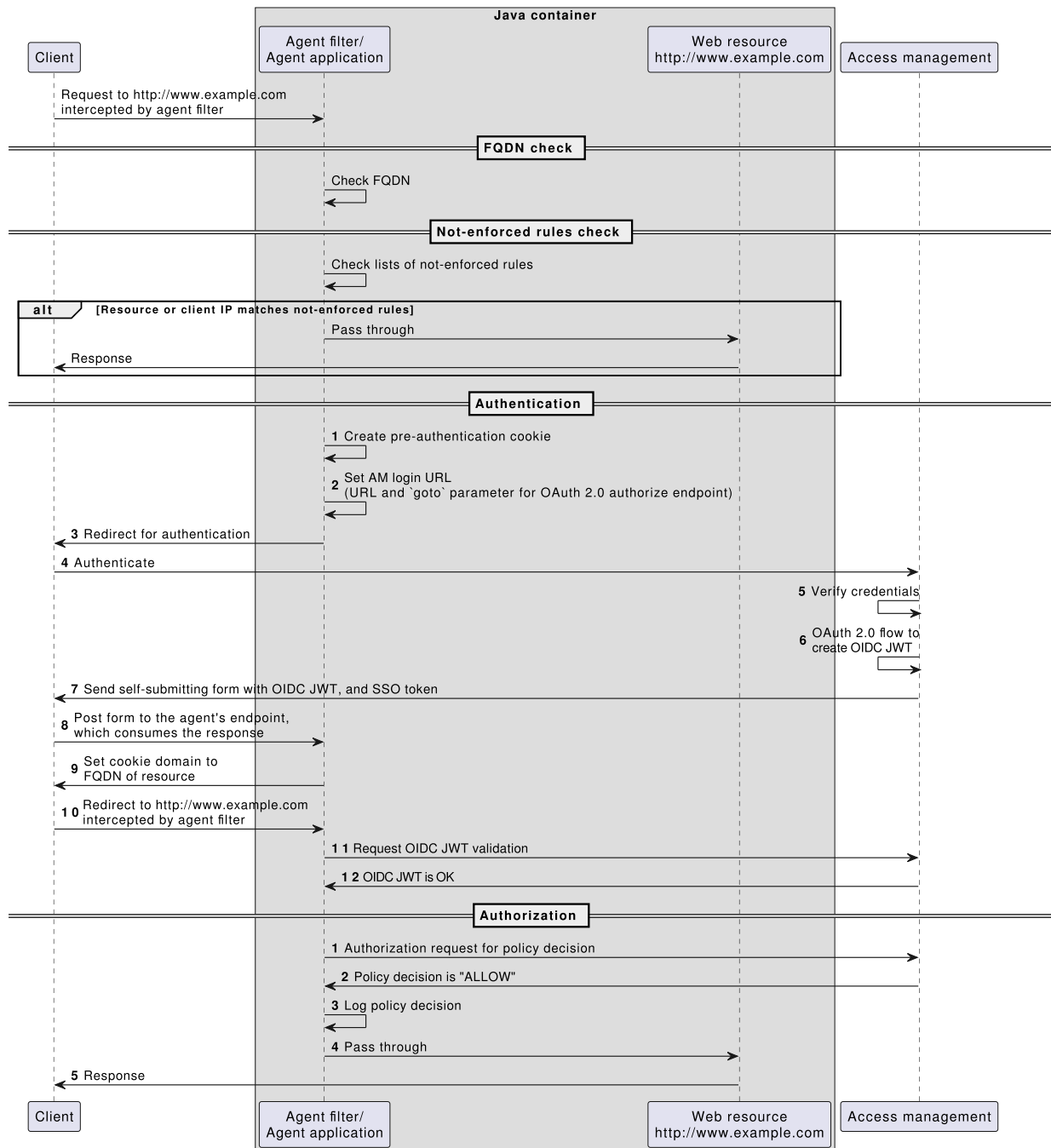
FQDN check

When FQDN checking is enabled, the agent can redirect requests to different domains, depending on the hostname of the request. For more information, see FQDN checking.

Not-enforced rules check

The agent evaluates whether the requested resource or the client IP address matches a not-enforced rule.

If 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.

For more information, see Not-enforced rules.

Authentication

1: 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. For added security, the pre-authentication cookie can be optionally be signed.

2: The agent sets the AM login URL, which includes the `goto` parameter for the OAuth 2.0 authorize endpoint, and

3: The agent redirects the client to log in to AM.

4-7: 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.

8: The client posts the self-submitting form to the agent endpoint, and the Java Agent consumes it.

9: The agent sets the cookie domain to the FQDN of the resource.

10: The client attempts to access the protected resource again, and the agent intercepts the request.

11: The agent contacts AM to validate the session contained in the OIDC JWT.

12: AM validates the session.

Authorization

1: The agent contacts AM's Policy Service, requesting a decision about whether the client is authorized to access the resource.

2: AM's Policy Service returns `ALLOW`.

3: The agent writes the policy decision to the audit log.

4: 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.

5: The client accesses the resource.

Cross-domain single sign-on

In Cross-Domain Single Sign-On (CDSSO), 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:
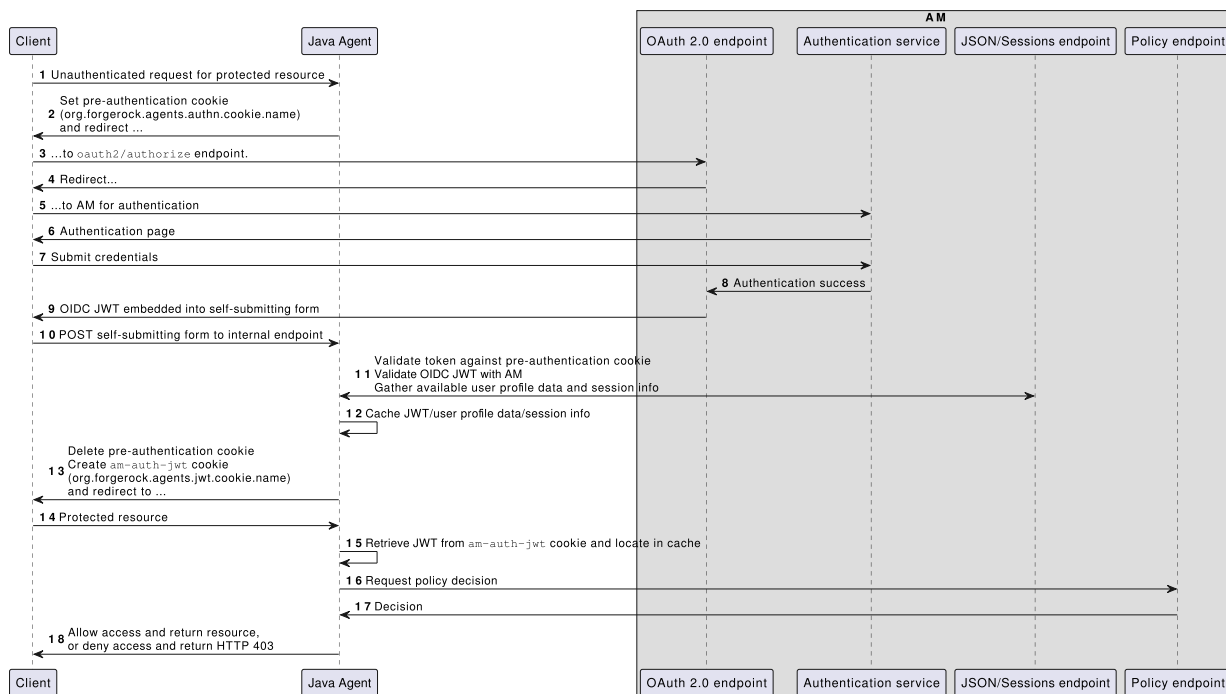
When the agent is in local configuration mode, configure the Authentication Redirect URI. When the agent is in remote configuration mode, the value is set by the agent configuration in AM.

For more information, see Single sign-on and Implement CDSSO in AM's *Authentication and SSO guide*.

Policy enforcement

This example sets up AM as a policy decision point for requests processed by Java Agent.

Before you start, install a Java Agent as described in the Installation guide, with the following values:

- AM server URL: `http://am.example.com:8080/am`

- Agent URL: `http://agent.example.com:80/app`

- Agent profile name: `java-agent`

- Agent profile realm: `/`

- Agent profile password: `/secure-directory/pwd.txt`

Enforce a policy decision from AM

1. Using the <u>ForgeRock Access Management docs</u> for information, log in to AM as an administrator, and make sure that you are managing the `/` realm.

2. Add a Java Agent profile:

   a. In the AM admin UI, select **Applications** > **Agents** > **Java**.

   b. Add an agent with the following values:

      - Agent ID: `java-agent`

      - Agent URL: `http://agent.example.com:80/app`

      - Server URL: `http://am.example.com:8080/am`

      - Password: `password`

3. Add a policy set and policy:

   a. In the AM admin UI, select 🔑 **Authorization** > **Policy Sets**, and add a policy set with the following values:

      - **Id** : `PEP`

      - **Resource Types** : `URL`

   b. In the policy set, add a policy with the following values:

      - **Name** : `PEP-policy`

      - **Resource Type** : `URL`

      - **Resources** : `*://*:*/*`

   c. On the **Actions** tab, add actions to allow HTTP `GET` and `POST`.

   d. On the **Subjects** tab, remove any default subject conditions, add a subject condition for all `Authenticated Users`.

4. Assign the new policy set to the agent profile:

   a. In the AM admin UI, Select **Applications** > **Agents** > **Java**, and select your agent.

   b. On the agent page, select the **AM Services** tab.

   c. Set **Policy Set** to `PEP`, and then click **Save**.

5. Test the setup:

   a. In the AM admin UI, select 🪪 **Identities** > ➕ Add Identity, and add a user with the following values:

      - **Username** : `demo`

      - **First name** : `demo`

      - **Last name** : `user`

- **Email Address** : demo@example.com
- **Password** : Ch4ng31t

    b. Log out of AM, and clear any cookies.

    c. Go to `http://agent.example.com:80/app`. The AM login page is displayed.

    d. Log in to AM as user `demo`, password `Ch4ng31t`, to access the web page protected by the Java Agent.

Retrieve advice or response attributes from policy decisions

When AM makes a policy decision, it communicates an entitlement to the agent, which can optionally include advice and response attributes.

When AM denies a request with advice, the agent uses the advice to take remedial action. For example, when AM denies a request because the authentication level is too low, it can send advice to increase the authentication level. The agent then prompts the user to reauthenticate at a higher level, for example, by using a one-time password.

When AM allows a request, it can include the following types of response attribute in the entitlement:

- Subject response attributes: Any LDAP user attribute configured for the identity store where AM looks up the user's profile. For more information, see Identity stores in AM's *Setup guide*.

- Static response attributes: Any key:value pair, for example, `FrequentFlyerStatus` : `gold`.

Depending on the value of Response Attribute Map, and Response Attribute Fetch Mode, the agent adds the listed attributes to HTTP headers, HTTP cookies, or request attributes in the response.

> This example builds on the example in Enforce a policy decision from AM. Set up and test that example first.
>
> 1. Configure subject response attributes and static response attributes in the AM policy you created earlier:
>
>   a. In the AM admin UI, select the `PEP-policy`, and go to the **Response Attributes** tab.
>
>   b. In the **SUBJECT ATTRIBUTES** frame, select one or more of the available attributes. For example, select `cn`.
>
>   c. In the **STATIC ATTRIBUTES** frame, add a response attribute pair. For example, add the following pair:
>
>   - **PROPERTY NAME**: `FrequentFlyerStatus`

- **PROPERTY VALUE**: gold

d. Click **Save Changes**.

2. In the AM admin UI, select the `java-agent` you created earlier.

   The agent must use the AM policy set and realm where the response attributes are configured.

   If the response attributes are not present in the policy decision from AM, the agent does not create the corresponding HTTP header or cookie.

3. In the **Application** tab, set **Response Attribute Fetch Mode** to select whether to map response attribute names to HTTP headers, HTTP cookies, or request attributes. For more information, refer to Response Attribute Fetch Mode.

4. In the **Response Attribute Map** field, map the subject response attributes you selected in AM:

   - **Key**: `cn`

   - **Value**: `CUSTOM-name`

     The name of the AM response attribute `cn` is mapped to an HTTP header, HTTP cookie, or request attribute called `CUSTOM-name`. The value is taken from the user profile.

     For more information, refer to Response Attribute Map.

5. In the **Response Attribute Map** field, map the static response attributes you added in AM:

   - **Key**: `FrequentFlyerStatus`

   - **Value**: `CUSTOM-flyer-status`

     The name of the AM response attribute `Frequent flyer status` is mapped to an HTTP header, HTTP cookie, or request attribute called `CUSTOM-flyer-status`. The value is `gold`.

     For more information, refer to Response Attribute Map.

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 temporarily, 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.

The data can be any POST content, such as HTML form data or a file upload.

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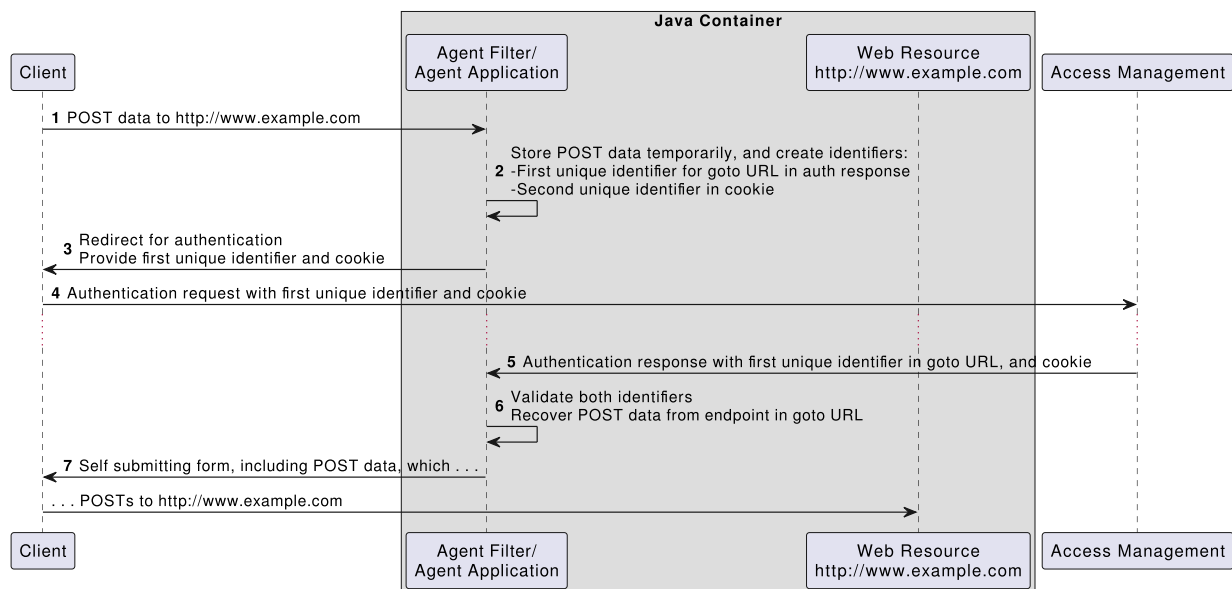
Java Agent guarantees the integrity of the data, and the authenticity of the client as follows:

1. An unauthenticated client requests a POST to a protected resource.

2. The agent stores the POST data temporarily, and then generates the following unique identifiers:

   ○ An identifier in the goto URL for the authentication response

   ○ 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 admin UI.

Security considerations for storing POST data in files

By default, POST data is stored in the in-memory cache. Consider the following points if you configure POST Data Preservation in Files or Cache to store POST data in the file system:

- Payloads from unauthenticated users are stored in the agent file system. If your threat evaluation does not accept this risk, store the data in the cache, or set POST Data Preservation in Files or Cache to `false`.

- Restrict access to the POST Data Preservation File Directory, to mitigate the risk of permissive access or leakage of personally identifiable information (PII).

- Limit the amount of stored POST data to mitigate the risk of DoS attacks, by configuring POST Data Preservation Storage Size or Max Entries in POST Data Preservation Storage.

- Remove expired POST data as soon as possible by configuring the POST Data Preservation Directory Sweep Interval.

- Identify threats in POST data before it is deleted, by making sure that Intrusion Detection Systems inspect the data within the time specified by POST Data Preservation Directory Sweep Interval.

## 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 recommends 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:
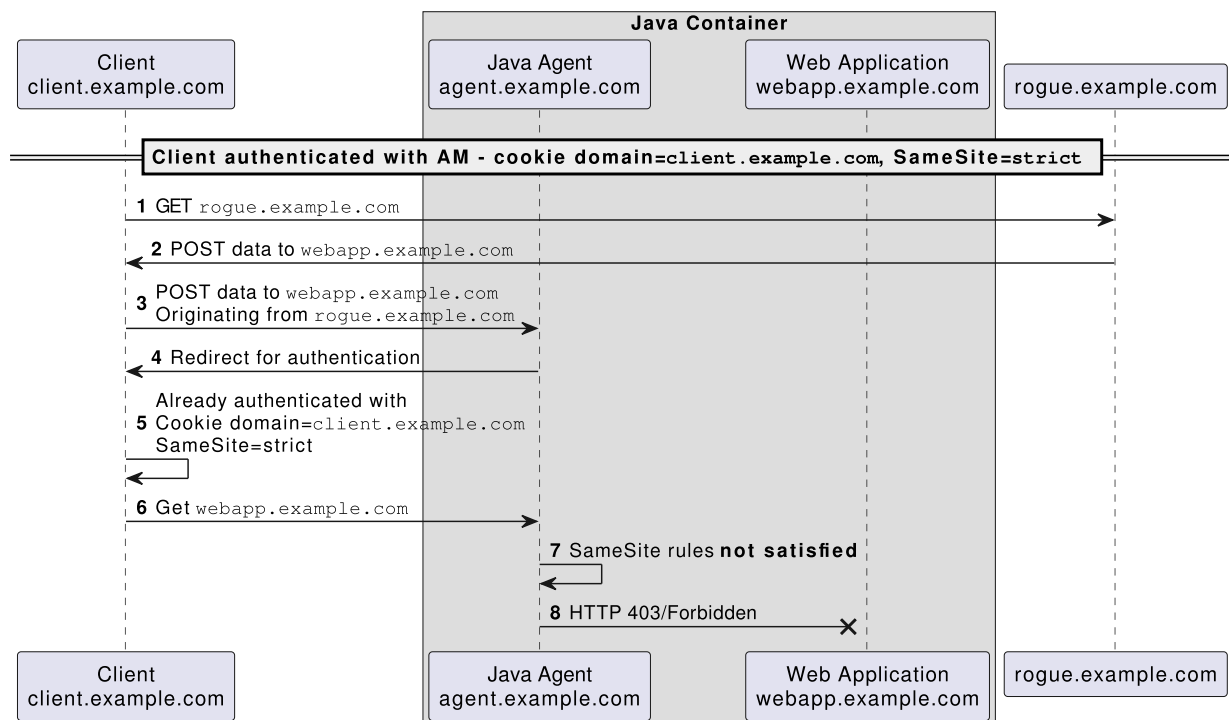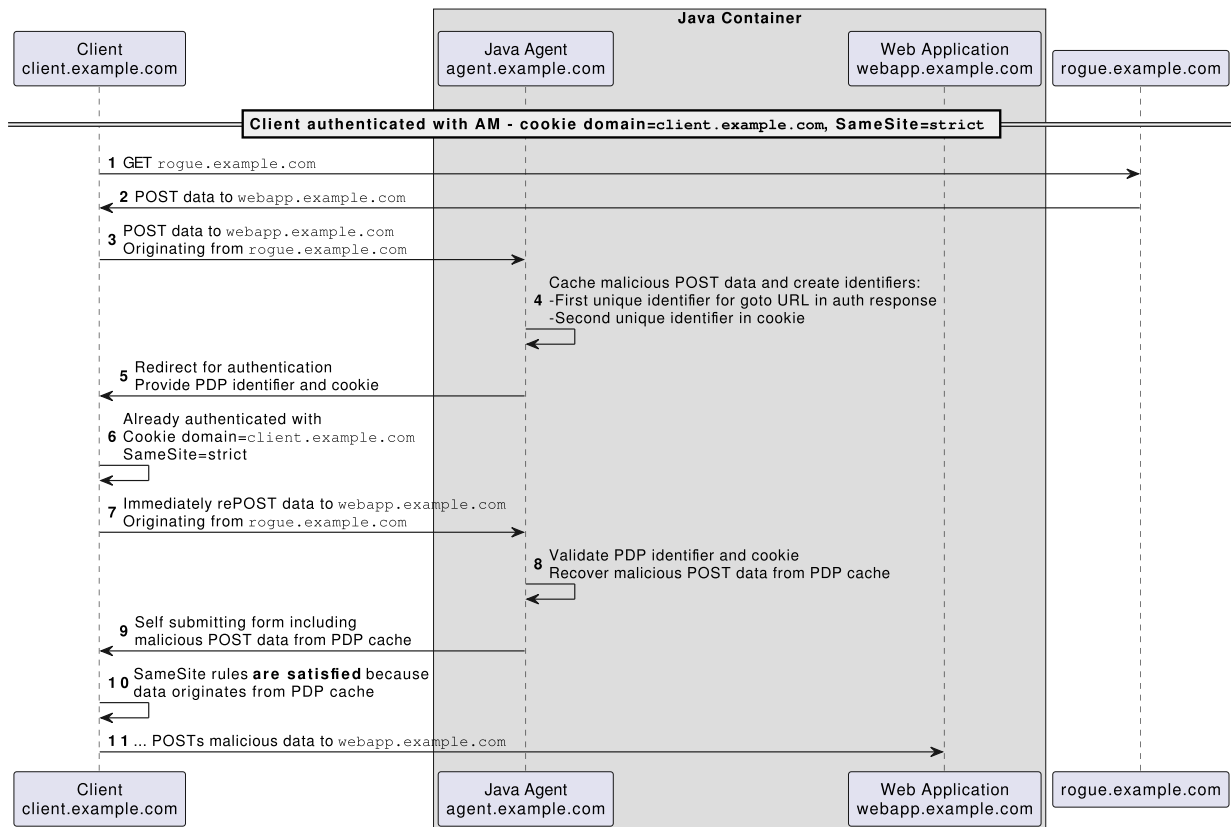

CSRF attack when POST data preservation is enabled

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:

Java Container containing: Java Agent (agent.example.com), Web Application (webapp.example.com)

Other participants: Client (client.example.com), rogue.example.com

Client authenticated with AM - cookie domain=client.example.com, SameSite=strict

1. GET `rogue.example.com`
2. POST data to `webapp.example.com`
3. POST data to `webapp.example.com` Originating from `rogue.example.com`
4. Cache malicious POST data and create identifiers: -First unique identifier for goto URL in auth response -Second unique identifier in cookie
5. Redirect for authentication Provide PDP identifier and cookie
6. Already authenticated with Cookie domain=`client.example.com` SameSite=strict
7. Immediately rePOST data to `webapp.example.com` Originating from `rogue.example.com`
8. Validate PDP identifier and cookie Recover malicious POST data from PDP cache
9. Self submitting form including malicious POST data from PDP cache
10. SameSite rules **are satisfied** because data originates from PDP cache
11. ... POSTs malicious data to `webapp.example.com`

Login redirect

When an unauthenticated user requests access to a protected resource, Java Agent redirects the user's browser to a login endpoint. The choice of the login endpoint, and the parameters it receives, is defined by the login redirect mode, default or custom.

Default login redirect

In default login redirect mode, the property Enable Custom Login Mode is always `false`. Depending on the configuration of login redirect properties, some endpoint parameters can be changed. For example, the agent can conditionally redirect a request to a specific realm or a different AM instance.

The `/oauth2/authorize` endpoint returns an OIDC ID token, and this is the only response the agent accepts.

Do **not** use default login redirect mode if session tokens for authentication and authorization are SSO tokens, even if you intend that the agent converts the SSO tokens into JWTs. Instead, use custom login redirect mode.

The following image shows the flow of data during a default login redirect:
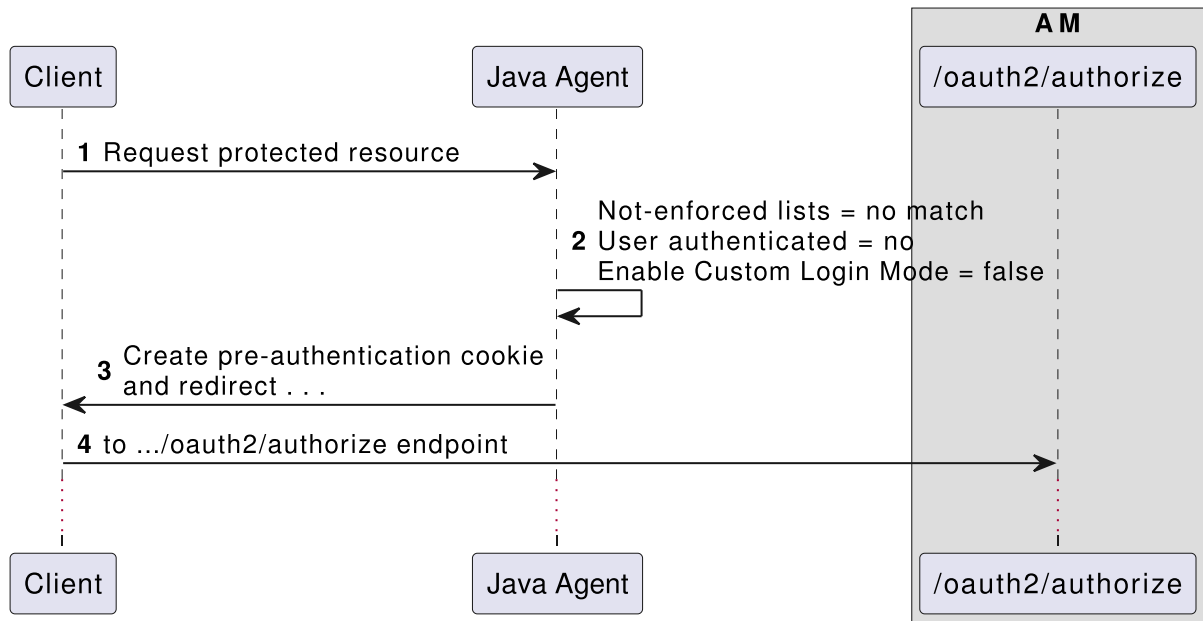
Figure 1. Data flow for default login redirect mode

Use the request domain to redirect login to a different realm

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

- Enable Custom Login Mode: Leave with the default value of `false`.

- AM Login URL List: Set to the URL of the login page, and specify the login realm as a parameter: `https://am.example.com:8443/am?realm=/myrealm`

The following image builds on figure 1, to configure AM Login URL List so that the agent redirects the user to log in to `myrealm` instead of the top-level realm.

Figure 2. Data flow for default login redirect mode, where the user authenticates to a subrealm

Use the request domain to redirect login to a subrealm

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 builds on figure 1, to configure OAuth Login URL List (`org.forgerock.agents.oauth.login.url.list`). Because the request is for a resource in `blue.example.com`, it is directed for authentication to the `blue` realm.

Figure 3. Data flow for default login redirect mode, where the user authenticates to a subrealm based on the request domain

Other requests are directed as follows:

- 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 yellow 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.

Use the request domain to redirect login to different endpoints
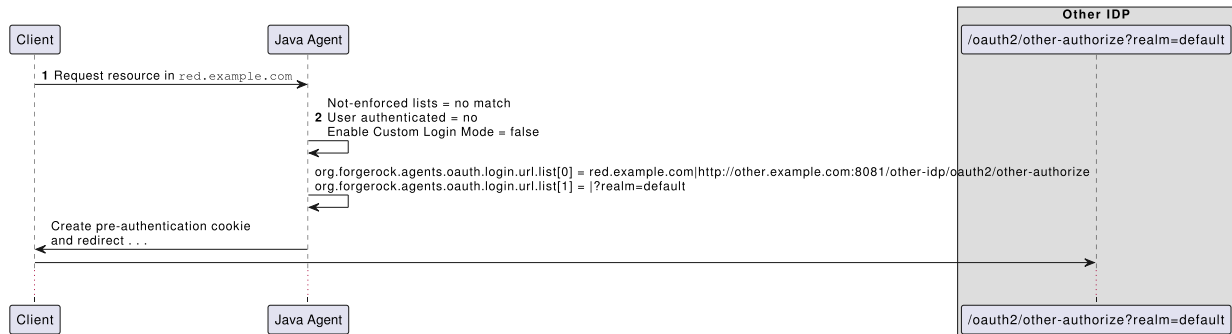
In default login redirect mode, the agent can redirect requests to any AM instance supporting the `/oauth2/authorize` endpoint.

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`.

- <u>OAuth Login URL List</u>: Map the request domain to the required OIDC 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 log in at the matched OIDC endpoint.

The following image builds on figure 1, to configure <u>OAuth Login URL List</u>. Because the request is for a resource in `red.example.com/yellow`, it is directed for authentication to a different IDP.

Figure 4. Data flow for default login redirect mode, where the user authenticates to an identity provider based on the request domain

Requests for a resource in an unmapped domain are passed to the AM `oauth2/authorize` endpoint, to log the user in to the specified default realm.

Custom login redirect

In custom login redirect mode, the agent is not confined to invoking a fixed endpoint in AM, but can redirect login anywhere. The agent handles JWTs or SSO tokens as session tokens for authentication and authorization.

Use custom login redirect mode for legacy deployments, where SSO tokens, instead of JWTs, are used for authentication and authorization. Otherwise, use <u>default login redirect</u> instead.

The property <u>Enable Custom Login Mode</u> is always `true`. Depending on the configuration of <u>login redirect properties</u>, the agent can:

- Convert SSO tokens into JWTs, through a direct "backdoor" call to AM
- Use caches to stop the SSO to JWT conversion from occurring more than once
- Leave SSO tokens unconverted

The following image shows the possible data flows for custom login redirect mode:
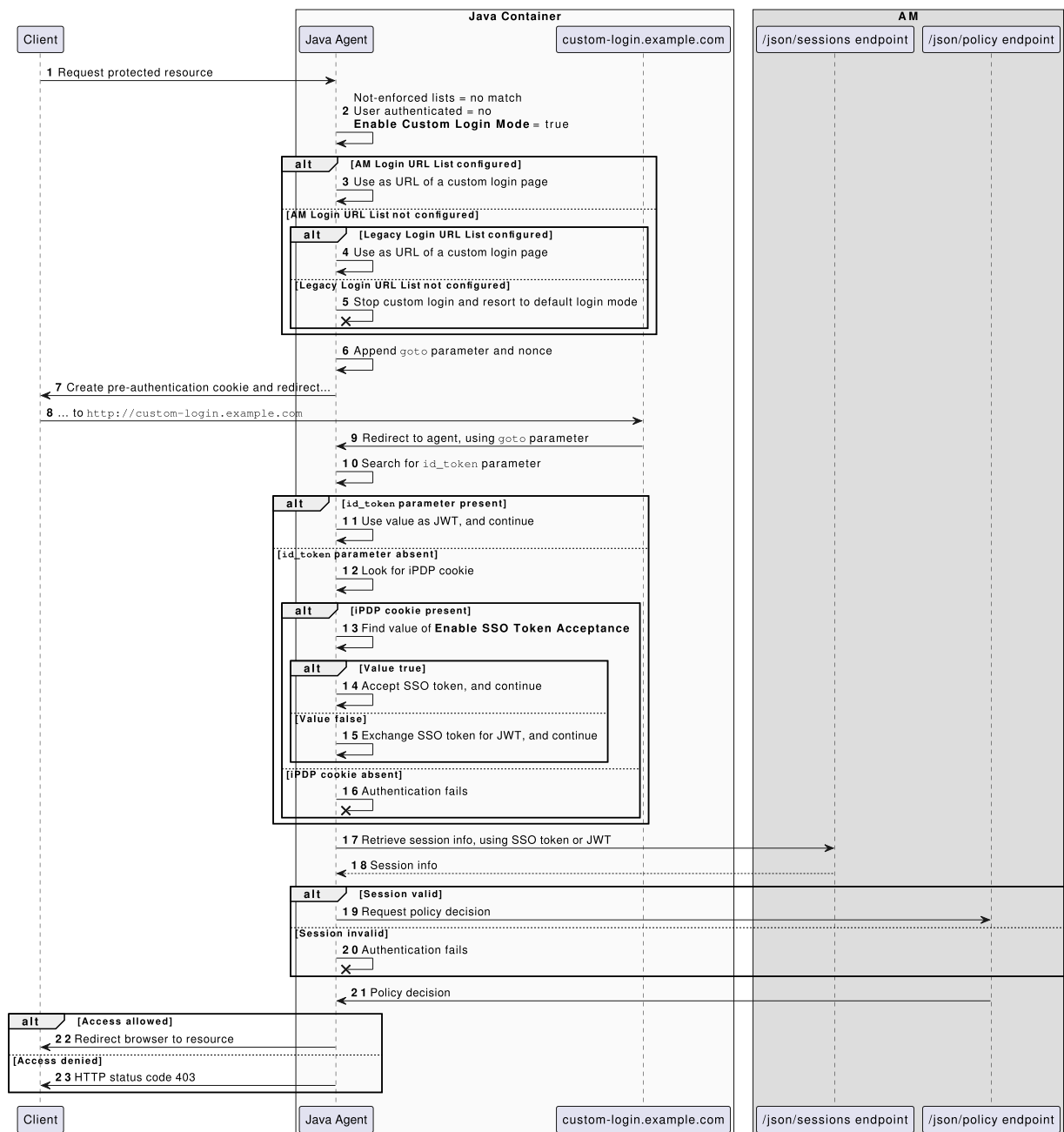
Java Container

Client | Java Agent | custom-login.example.com

AM

/json/sessions endpoint | /json/policy endpoint

1 Request protected resource

Not-enforced lists = no match
2 User authenticated = no
Enable Custom Login Mode = true

alt [AM Login URL List configured]
3 Use as URL of a custom login page

[AM Login URL List not configured]

alt [Legacy Login URL List configured]
4 Use as URL of a custom login page

[Legacy Login URL List not configured]
5 Stop custom login and resort to default login mode

6 Append `goto` parameter and nonce

7 Create pre-authentication cookie and redirect...

8 ... to `http://custom-login.example.com`

9 Redirect to agent, using `goto` parameter

10 Search for `id_token` parameter

alt [`id_token` parameter present]
11 Use value as JWT, and continue

[`id_token` parameter absent]
12 Look for iPDP cookie

alt [iPDP cookie present]
13 Find value of **Enable SSO Token Acceptance**

alt [Value true]
14 Accept SSO token, and continue

[Value false]
15 Exchange SSO token for JWT, and continue

[iPDP cookie absent]
16 Authentication fails

17 Retrieve session info, using SSO token or JWT

18 Session info

alt [Session valid]
19 Request policy decision

[Session invalid]
20 Authentication fails

21 Policy decision

alt [Access allowed]
22 Redirect browser to resource

[Access denied]
23 HTTP status code 403

Client | Java Agent | custom-login.example.com | /json/sessions endpoint | /json/policy endpoint

Figure 5. Data flow for customized login redirect

Redirect login to a custom URL configured in AM

AM's **OAuth2 Provider** service can be configured to use a custom URL to handle login, to override the default AM login page. When a custom login page is configured in AM, configure the agent to ensure that it redirects the login to that page.

1. In the AM admin UI, go to **Services** > **OAuth2 Provider** > **Advanced** > **Custom Login URL Template**, and note the custom URL.

2. Go to **Applications** > **Agents** > **Java**, and select your Java Agent.

3. On the **AM Services** tab set the following properties:

   ○ Enable Custom Login Mode: Set to on

Redirect login to AM behind a firewall

When login must be completed in a network where AM is behind a firewall, set Public AM URL to a proxy which can access AM.

Limit the number of allowed redirect attempts

To mitigate the risk of infinite redirection loops, limit the number of redirects allowed for a browser session. After this number, the agent blocks the request.

Configure Redirect Attempt Limit, to specify a non-zero value. For example, if the limit is set to three, then the agent blocks the request on the fourth redirect.

Logout

This section describes how to trigger a logout based on the properties of a request, and how to redirect users after logout to a specified logout resource.

The URL used to trigger logout can be the agent's own URL, or one overridden by the configuration. The logout URL is expected to register the session destruction with AM.

The agent maintains the user realm for each session, either by obtaining the realm info from the JWT, or by calling the `sessioninfo` endpoint (when SSO tokens are used). When the user logs out, the stored realm is passed to the logout endpoint automatically.

AM manages session cookies as follows, and the agent is responsible for destroying the cookies:

- From AM 7, AM places the session cookie in the `Authorization` header, prefixed with `X-Requester-Token`.
- Before AM 7, AM places the session cookie in the HTTP parameter `requester`.

If Convert SSO Tokens Into OIDC JWTs is `true`, the logout URL is invoked twice—once with the JWT, and again with the SSO token. If Enable SSO Token Acceptance is `true`, the logout URL can be invoked only by an SSO token.

Configure logout with the properties described in Logout.

Trigger logout with a URL

Set the property Logout URI Map to specify a URL to trigger logout. When the URL is invoked, the agent kills the current session by invoking the AM REST logout endpoint or

the endpoint configured by Conditional Logout URL List.

The URL is a dummy URL. Even if a resource exists at the URL, it is never accessed.

Log out of a specific web application

The following example triggers a logout from an application called `bank` , when the URL `http://app.example.com:80/mywebapp/bank/log-me-out` is invoked:

```
org.forgerock.agents.logout.endpoint.map[bank]=/bank/log-me-out
```

When a web application is specified, it must exist and the agent must have access to it. If the `bank` application in the above example doesn't exist, the web container throws an error.

Log out of all web applications

If a web application is not specified, the current sessions are killed for all web applications. The following examples trigger a logout from any application when the specified URL is invoked:

```
org.forgerock.agents.logout.endpoint.map=/agentapp/log-me-out
```

The agent must be able to access the context for the URL. For example, unless the agent is deployed in the root context, the following configuration fails:

```
org.forgerock.agents.logout.endpoint.map=/dummy-logout
```

Trigger logout with a parameter

Set the property Logout Request Parameter Map to specify a URL parameter to trigger logout. The agent searches every incoming request for the parameter. When the agent detects the parameter, it invokes AM to kill the current session for the specified web application.

To speed up the search for a logout parameter, set the property Enable Logout Introspection to `true` .

Log out of a specific web application

The following example triggers a logout from an application called `bank` when the request URL contains the parameter `log-out` :

```
org.forgerock.agents.logout.request.param.map[bank]=log-out
```

The request URL must contain the `log-out` parameter, but does not need to assign a value to the parameter. The following request URLs would trigger a logout for the previous configuration:

```
http://am.example.com:8080/protectedapp/index.html?log-out
http://am.example.com:8080/examples/index.html?examplelog-out=
```

Log out of all web applications

If a web application is not specified, the current sessions are killed for all web applications. The following example triggers a logout from any application when the request URL contains the parameter `logout`:

```
org.forgerock.agents.logout.request.param.map=logout
```

Conditionally log out to different URLs

Set the property Conditional Logout URL List to define a URL to which the agent can conditionally direct the user on logout.

If Conditional Logout URL List is set to a URL that does not perform a REST logout to AM, set Always invalidate sessions to `true`. The agent additionally invokes the AM REST logout endpoint to invalidate the session.

Configure one or more conditions. The request URL is compared to each condition in the list until the closest match is found. Conditions are evaluated by order of length, starting with the longest, irrespective of their order in the list.

In the following example, `example.com/path` is evaluated before `example.com`; the default condition is the shortest, and is evaluated last:

```
org.forgerock.agents.conditional.logout.url.list[0]=example.com|?
additional=value
org.forgerock.agents.conditional.logout.url.list[1]=example.com/p
ath|?one=red&two=green&three=blue
org.forgerock.agents.conditional.logout.url.list[2]=mybank.com|ht
tp://mybank.com/myapp/logout?param=override
org.forgerock.agents.conditional.logout.url.list[3]=|?alpha=beta
```

Using the above configuration, consider the following evaluations:

Request URL	Action
`http://example.com:9010/path/index.html`	The following parameter name:value pairs are added to the logout URL: `one:red`, `two:green`, and `three=blue`
`http://example.com:9010:/path/public/index.html`	
`http://example.com:9010:/index.html`	The following parameter name:value pair is added to the logout URL: `additional:value`
`https://mybank.com:443/path/index.html`	`http://mybank.com/myapp/logout` is used for logout, overriding the AM logout REST endpoint that the agent would use by default. The administrator is responsible for making sure that the overriding URL kills all tokens associated with login, but is not responsible for removing cookies containing either JWTs or SSO tokens.
Any URL that does not match on of the other conditions	Parameter `alpha:beta` added to logout URL

Redirect logout to a landing page

Set the property Logout Entry URI Map to redirect users to a specified resource after logout using an endpoint defined in Logout URI Map or Logout Request Parameter Map.

The specified resource can be an HTML page or JSP file. It is automatically added to the not-enforced list, so that it can be accessed without authentication.

Configure a logout landing page for a specific web application

The following example directs requests to the `bank` application to `logout-page.html`, after logout:

```
org.forgerock.agents.logout.goto.map[bank]=/banking-app/logout-page.html
```

Configure a logout landing page for all web applications

To redirect requests for any web application, leave the web application name field empty, and set the logout URI as a specific URL. The following example directs all requests to

goodbye.html after logout:

```
org.forgerock.agents.logout.goto.map=/agentapp/goodbye.html
```

Not-enforced rules

When an agent is configured to protect a web application, it protects **every** resource in that application. Each access to any resource incurs the overhead of a policy evaluation request to AM; access 100 resources, and you incur 100 requests to AM, with all of their associated network overhead.
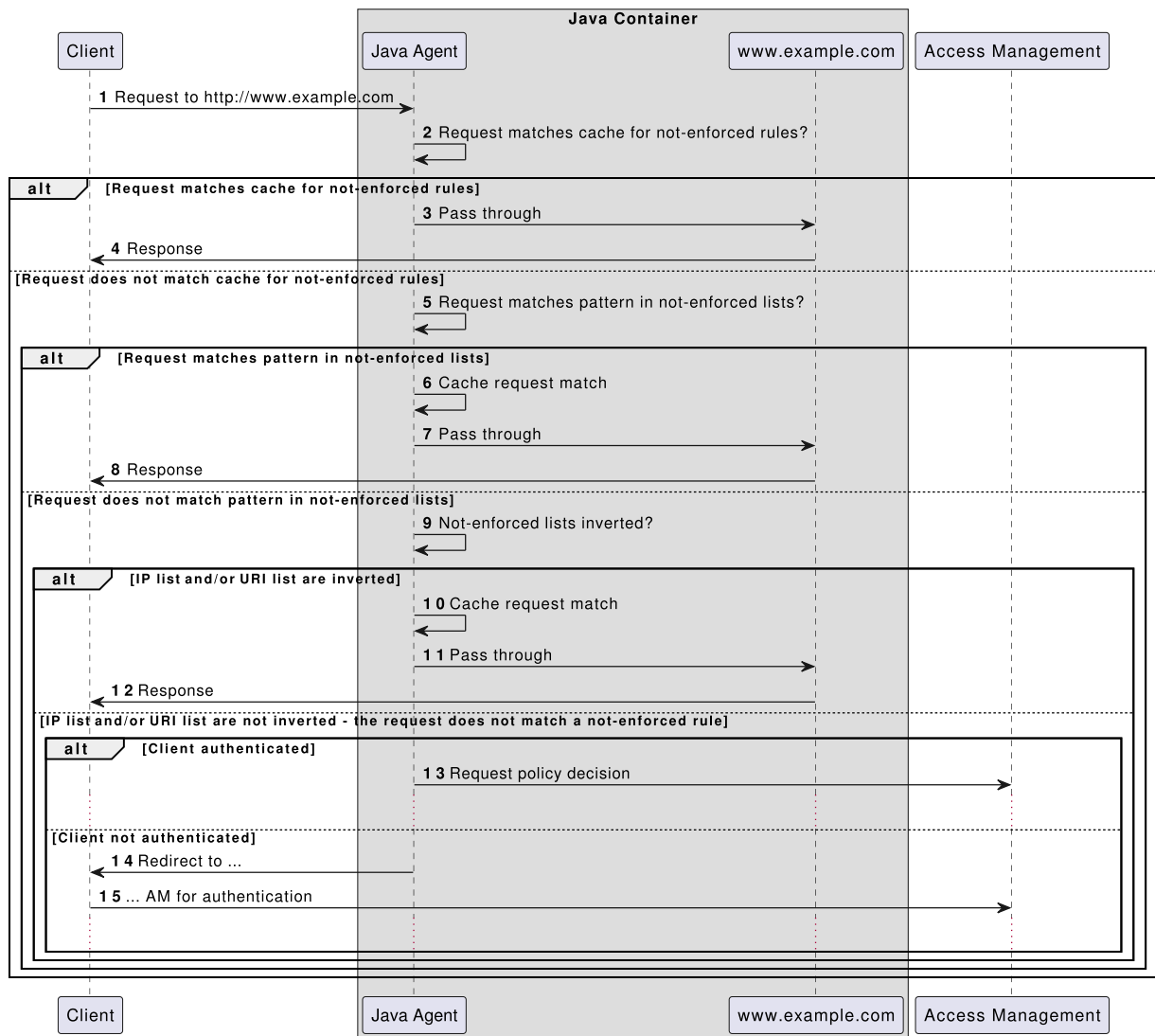
Some resources, such as the "public" directory of a web application, contain data that is not sensitive. It can be accessed by any, authenticated or unauthenticated, clients. The agent uses lists of *not-enforced rules* to identify these resources in the web application.

The agent matches incoming requests to the lists of not-enforced rules. When a request matches a not-enforced rule, the agent bypasses the call to AM:

- 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.

Use not-enforced rules to reduce the number of unnecessary calls to AM, and therefore improve the performance and speed of your application.

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:

Java Container

Client → Java Agent — **1** Request to http://www.example.com

Java Agent → Java Agent — **2** Request matches cache for not-enforced rules?

alt [Request matches cache for not-enforced rules]

Java Agent → www.example.com — **3** Pass through

www.example.com → Client — **4** Response

[Request does not match cache for not-enforced rules]

Java Agent → Java Agent — **5** Request matches pattern in not-enforced lists?

alt [Request matches pattern in not-enforced lists]

Java Agent → Java Agent — **6** Cache request match

Java Agent → www.example.com — **7** Pass through

www.example.com → Client — **8** Response

[Request does not match pattern in not-enforced lists]

Java Agent → Java Agent — **9** Not-enforced lists inverted?

alt [IP list and/or URI list are inverted]

Java Agent → Java Agent — **10** Cache request match

Java Agent → www.example.com — **11** Pass through

www.example.com → Client — **12** Response

[IP list and/or URI list are not inverted - the request does not match a not-enforced rule]

alt [Client authenticated]

Java Agent → Access Management — **13** Request policy decision

[Client not authenticated]

Java Agent → Client — **14** Redirect to ...

Client → Access Management — **15** ... AM for authentication

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 properties listed in Not-enforced rules in the *Properties reference*, or on the `Application` tab of the AM admin UI. Configure the following lists of not-enforced rules:

Not-enforced URI rules

Rules in Not-Enforced URIs 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

Rules in Not-Enforced Client IP List 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 the following order:

Order	Rule type	Rule includes requirements for Cookie values or Header values	Rule uses the `DENY` keyword to Deny access
1	Compound	Yes	Yes
2	Compound	Yes	No
3	Compound	No	Yes
4	Compound	No	No

Order	Rule type	Rule includes requirements for Cookie values or Header values	Rule uses the `DENY` keyword to Deny access
5	IP	Yes	Yes
6	IP	Yes	No
7	IP	No	Yes
8	IP	No	No
9	URL	Yes	Yes
10	URL	Yes	No
11	URL	No	Yes
12	URL	No	No

Conventions for not-enforced rules

Use the following conventions to define not-enforced URI rules and not-enforced IP rules.

Path normalization

Java Agent normalizes resource paths *before* applying not-enforced rules.

This affects the following sequences, for example:

- `%2F` becomes `/`
- `%5C` becomes `/`

Invert rules

Invert specific rules

> **IMPORTANT**
>
> The `NOT` keyword is overridden by the DENY keyword. Refer to Deny access.

Invert any rule in a 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 enforces authentication and requests policy evaluations for any request from the network specified by the `192.168.1.0/24` CIDR notation:

```
NOT 192.168.1.0/24
```

Invert all rules

> **IMPORTANT**
>
> For security considerations, do not invert all rules. Instead, ForgeRock recommends using the NOT keyword to invert specific rules.

> **IMPORTANT**
>
> The `NOT` keyword and the properties in this section are overridden by the `DENY` keyword. Refer to <u>Deny access</u>.

Invert all rules by setting <u>Invert Not-Enforced IPs</u> or <u>Invert Not-Enforced URIs</u> to `true`.

Wildcards

For more information about using wildcards, see <u>Wildcards</u>.

Note that trailing forward slashes are not recognized as part of a resource name. Therefore, `/images//`, `/images/`, and `/images` are equivalent.

Multi-level wildcard (*)

The following list summarizes the behavior of the multi-level wildcard (*):

- Matches zero or more occurrences of any character except for the question mark (?).
- Spans multiple levels in a URL.
- Cannot be escaped. Therefore, the backslash character (\ ) or other characters cannot be used to escape the asterisk, as such `\*`.
- * Cannot be used in the same rule as the one-level wildcard (`-*-`) or a regular expression.

The following table gives examples of the multi-level wildcard `*` in rules defined in <u>Not-Enforced Client IP List</u>:

Multi-level wildcard for not-enforced IP rules

Rule	Matches request IP	Does not match request IP
`192.168.1.*`	`192.168.1.0` `192.168.1.0/24`	`192.168.0.1`

The following table gives examples of the multi-level wildcard `*` in rules defined in Not-Enforced URIs:

Multi-level wildcard for not-enforced URI rules

Rule	Matches request URL	Does not match request URL
`http://A-examp.com:8080/*`	`http://A-examp.com:8080/` `http://A-examp.com:8080/index.html` `http://A-examp.com:8080/x.gif`	`http://B-examp.com:8080/` `http://A-examp.com:8090/index.html` `http://A-examp.com:8080/a?b=1`
`http://A-examp.com:8080/*.html`	`http://A-examp.com:8080/index.html` `http://A-examp.com:8080/pub/ab.html` `http://A-examp.com:8080/pri/xy.html`	`http://A-examp.com/index.html` `http://A-examp.com:8080/x.gif` `http://B-examp.com/index.html`
`http://A-examp.com:8080/*/ab`	`http://A-examp.com:8080/pri/xy/ab/xy/ab` `http://A-examp.com:8080/xy/ab`	`http://A-examp.com/ab` `http://A-examp.com/ab.html` `http://B-examp.com:8080/ab`

Rule	Matches request URL	Does not match request URL

`http://A-examp.com:8080/ab/*/de`	`http://A-examp.com:8080/ab/123/de`
	
	`http://A-examp.com:8080/ab/ab/de`
	
	`http://A-examp.com:8080/ab/de/ab/de`
		`http://A-examp.com:8080/ab/de`
		
		`http://A-examp.com:8090/ab/de`
		
		`http://B-examp.com:8080/ab/de/ab/de`

One-level wildcard (--)*

The following list summarizes the behavior of the one-level wildcard (-*-):

- Matches zero or more occurrences of any character except for the forward slash (/) and the question mark (?).

- Does not span multiple levels in a URL.

- Cannot be escaped. Therefore, the backslash character (\) or other characters cannot be used to escape the hyphen-asterisk-hyphen, like this \-*- .

- Cannot be used in the same rule as the multi-level wildcard (*) or a regular expression.

The following table gives examples of the one-level wildcard -*- in rules defined in Not-Enforced URIs:

One-level wildcard for not-enforced URI rules

Rule	Matches request URL	Does not match request URL

`http://A-examp.com:8080/b/-*-`	`http://A-examp.com:8080/b/`
	
	`http://A-examp.com:8080/b/cd`
	
	`http://A-examp.com:8080/b/cd/`
		`http://A-examp.com:8080/b`
		
		`http://A-examp.com:8080/b/c?d=e`
		
		`http://A-examp.com:8080/b/cd/e`
		
		`http://A-examp.com:8090/b/`

Rule	Matches request URL	Does not match request URL

`http://A-examp.com:8080/b/-*-/f`	`http://A-examp.com:8080/b/c/f`
	
	`http://A-examp.com:8080/b/cde/f`
		`http://A-examp.com:8080/b/c/e/f`
		
		`http://A-examp.com:8080/f/`

`http://A-examp.com:8080/b/c-*-/f`	`http://A-examp.com:8080/b/cde/f`
	
	`http://A-examp.com:8080/b/cd/f`
	
	`http://A-examp.com:8080/b/c/f`
		`http://A-examp.com:8080/b/c/e/f`
		
		`http://A-examp.com:8080/b/c/`
		
		`http://A-examp.com:8080/b/c/fg`

Multiple wildcards

When multiple wildcards are included in the same rule of a <u>Not-Enforced URIs</u>, the agent matches the parameters in any order that they appear in a resource URI.

For example, the following rule applies to any resource URI that contains a `member_level` and `location` query parameter, in any order:

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

The following requests would be not-enforced:

```
https://www.example.com/customers/default.jsp?
member_level=silver&location=fr
https://www.example.com/customers/default.jsp?
location=es&member_level=silver
https://www.example.com/customers/default.jsp?
location=uk&vip=true&member_level=gold
```

Deny access

Use the `DENY` keyword with a regular expression or classic pattern to prevent access either to a resource or from an IP address. If the request matches the resource path or IP address, Java Agent denies access with an HTTP 403 Forbidden status.

When the `DENY` keyword is used, access to a resource and/or access from an IP address is *always* denied. If the `NOT` keyword or the following properties are used with the `DENY`

keyword, they are ignored: <u>Invert Not-Enforced IPs</u> or <u>Invert Not-Enforced URIs</u>.

Consider the following examples:

- Deny access to requests from the specified IP address:

```
DENY 155.251.79.32
```

- Deny access to all .jpg files:

```
DENY /*.jpg
```

  Note here that the use of `DENY` causes `NOT` to be ignored:

```
NOT,DENY /*.jpg
```

- `DENY` access to incoming URLs containing dubious characters, such as percent characters remaining in the normalized path. Java Agent converts `%2F` to `/` and `%5C` to `\` before applying not-enforced rules:

```
DENY /*%*
```

Regular expressions

<div>
IMPORTANT

The Java Agent uses regular expression matching from the JDK. Make sure your expressions are evaluated in a way that is consistent with this.
</div>

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

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

```
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.*
```

In the following example, the agent does not enforce authentication or request policy evaluations for GET requests to `/public`, but does 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/([^/])+/.*\.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. In the following examples, the agent enforces authentication and requests policy evaluations 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 require the incoming request to have 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

- 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 with the case-insensitive name `login_result`, and case-insensitive value `valid`:

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

Because the search is case-insensitive, the following example is equivalent:

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

In the following example, the agent does not enforce authentication or request policy evaluations for requests to `192.168.*`, when the request contains a cookie with the case-sensitive 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.

In the following example, the agent does not enforce authentication or request policy evaluations for GET, POST, and PUT requests to the `/other/records/` folder, when the request contains a cookie with the case-sensitive name `internal`, and a case-insensitive value that ends with `.*ID`:

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

In the following example, the agent does not enforce authentication or request policy evaluations for `GET`, `POST`, and `PUT` HTTP requests from the client IP range `192.168.*`, when the request contains a cookie with the case-sensitive name `internal`, and a case-sensitive value that ends with `.*ID`:

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

Header values

Use the following syntax to require the incoming request to have 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.

In the following example, the agent does not enforce authentication or request policy evaluations for access to .txt files in `/yearly/2021/` when the request contains a header with the case-sensitive name `ID`, and a case-insensitive value `validated`

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

In the following example, the agent does not enforce authentication or request policy evaluations for access to the IP range `192.168.*` when the request contains a header with the case-sensitive name `ID`, and a 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.

In the following example, the agent does not enforce authentication or request policy evaluations for `GET`, `POST`, and `PUT` requests to HTML resources in the `/other/records/` folder when the request contains a header with the case-insensitive name `internal`, and a case-insensitive value that ends with `.*ID`:

```
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 <u>Not-Enforced Client IP List</u> or <u>Not-Enforced URIs</u>, 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 authentication or request policy evaluations for 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 authentication or request policy evaluations for `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 enforces authentication and requests policy evaluations for 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\/([^/])+\.*\.jpg
  ```

  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.

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 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 agent configuration properties, see <u>Continuous security</u>. For information about server-side authorization scripts, see <u>Scripting a policy condition</u> in AM's *Authorization guide*.

Environment maps with customizable keys

In Java Agent, use the continuous security properties <u>Client Hostname Header</u> and <u>Client IP Address Header</u> 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 <u>Client IP Address Header</u> 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 <u>Client Hostname Header</u> 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=77xe99f4zqi1l99z
myUser-Agent=Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0)
like Gecko
```

Environment maps with fixed keys

In Java Agent, use the following properties to configure an environment map with fixed keys:

- GET Parameter List for URL Policy Env
- POST Parameter List for URL Policy Env
- JSession Parameter List for URL Policy Env

Caching

Java Agent allocates memory from the Java heap space in the web container to the caches described in this section.

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
Client logs out from AM (and session notifications are enabled)	Session and policy decisions related to the session

Event	What is invalidated?
Policy decision reaches the expiration time specified by Policy Cache TTL	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 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.

Attribute fetch modes

For information about properties to configure attribute fetching, see Attributes.

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 using 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`.

FQDN checking

When FQDN checking is enabled, the agent can redirect requests to different domains,
depending on the hostname of the request. Use this feature in environments where the
request hostname can be virtual, invalid, or partial.

FQDN checking requires Enable FQDN Checking to be `true`, Default FQDN to be set to a
suitable value, and optionally, FQDN Map to be set to suitable default FQDN.

When FQDN Map is configured, the agent maintains the following maps:

- Map 1:

    - Key: Incoming hostname without wildcards.

    - Value: Outgoing hostname.

- Map 2:

    - Key: Incoming hostname with wildcards `*` and `?`.

    - Value: Outgoing hostname.

Map keys are case insensitive. Incoming hostnames are converted to lowercase before
the agent maps them, and the agent automatically converts uppercase keys and values to
lowercase before mapping.

The agent maps FQDNs as follows:

1. Searches map 1 for the incoming hostname. If there is a match, the agent redirects
   the request to the mapped value.

2. Searches map 2 for a pattern that matches the incoming hostname, iterating through
   the entries in random order. If there is a match, the agent redirects the request to the

mapped value.

3. Redirects the request to the value in <u>Default FQDN</u>.

Examples

The following example configuration and requests illustrate how the agent checks and remaps FQDNs:

Configuration

- <u>Enable FQDN Checking</u>: `org.forgerock.agents.fqdn.check.enabled=true`

- <u>Default FQDN</u>: `org.forgerock.agents.fqdn.default=agent.defaulttest.me`

- <u>FQDN Map</u>:

    - Map 1

      ```
      org.forgerock.agents.fqdn.map[agent]=agent.localtest.me
      org.forgerock.agents.fqdn.map[agent.virtualtest.me]=virtual-
      host.localtest.me
      ```

    - Map 2

      ```
      org.forgerock.agents.fqdn.map[agent-
      *.localtest.me]=agent.localtest.me
      ```

Example requests

- `https://agent.localtest.me/app` : Does not match any mapping, so the agent redirects it to the default FQDN `https://agent.defaulttest.me/app` .

- `https://agent/app` : The request URL matches the first mapping in map 1, so the agent redirects it to `https://agent.localtest.me/app` .

- `https://AGENT/app` : The request URL matches the first mapping in map 1, because incoming hostnames are converted to lower-case before the agent maps them. The agent redirects the request to `https://agent.localtest.me/app` .

- `https://agent.virtualtest.me/app` : The request URL matches the second mapping in map 1, so the agent redirects it to the virtual host `https://virtual-host.localtest.me/app` .

- `https://agent-123.localtest.me/app` : The request URL matches the mapping in map 2, so the agent redirects it to `https://agent.localtest.me/app` .

Cookie reset

Java Agent can reset cookies in its own domain before redirecting the client for login, and when the client logs out.

Pre-authentication cookies are reset automatically after successful authentication. Authentication cookies are reset automatically on logout. This section describes how to manage reset of other cookies.

To enable cookie reset, set <u>Cookie Reset</u> to `true`. The agent resets the cookies in the response before redirecting the client for login, and when the client logs out.

To reset specific cookies, add them to the list in <u>Reset Cookie List</u>. The agent searches for the cookie name using a case-sensitive search. If it finds a match, the cookie is reset. Otherwise, the agent searches again using a case-insensitive search. If it then finds a match, the cookie is reset and a warning is issued to the logs.

When profile or session attributes are stored in cookies (either <u>Profile Attribute Fetch Mode</u> or <u>Session Attribute Fetch Mode</u> has the value `HTTP_COOKIE`), cookie reset is enabled automatically and cannot be disabled. The agent resets the profile and session attributes cookies and the cookies in the <u>Reset Cookie List</u>.

When response attribute are stored in cookies (<u>Response Attribute Fetch Mode</u> has the value `HTTP_COOKIE`), the agent does not reset them automatically. To prevent a build up of response attribute cookies, consider adding them to the <u>Reset Cookie List</u>.

```
org.forgerock.agents.cookie.reset.name.list[0]=response-
attribute-cookie-name1
org.forgerock.agents.cookie.reset.name.list[1]=response-
attribute-cookie-name2
```

To specify the paths for which cookies named in <u>Reset Cookie List</u> are used after reset, set the <u>Reset Cookie Path Map</u>.

Consider enabling cookie reset when the agent is deployed with parallel authentication mechanisms. Resetting cookies from one authentication mechanism before redirecting clients to log in with another mechanism helps prevent issues on the new login site.

Authentication failure

When a client does not present a valid SSO token with a request, Java Agent redirects the client to login. If the client then fails to authenticate, by default, the agent takes the following steps:

1. Redirects the request to the URL defined by <u>Authentication Fail URL</u>.
2. If that property is not set, redirects the request to the URL defined by <u>Goto URL</u>.
3. If neither property is set, returns an HTTP 400.

To limit the amount of information available to malicious users, by default, the agent returns an HTTP 400 for all authentication failures, regardless of the reason.

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

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

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

Manage notifications for authentication failure

An HTTP 400 message is not always helpful for debugging the agent flow or when another web application depends on the error message. To change the way the agent responds to authentication failure, configure the following properties:

- Authentication Fail URL, to redirect the uses to a specific URL or URI. Use this property to control the message the agent displays to the client.

- Authentication Fail Reason Parameter Name, to send the reason for authentication failure in a named query parameter.

- Authentication Fail Reason Parameter Value Map, to map the reason for authentication failure. Use this property to hide the reason for authentication failure from malicious users, or to map it to something that is meaningful inside your organization.

Limit the number of failed login attempts

To mitigate the risk of brute force attacks, limit the number of failed login attempts that are allowed during a browser session. After this number, the agent blocks requests from the user.

Configure Login Attempt Limit, to specify a non-zero value. For example, if the limit is three, then the agent blocks the fourth and subsequent login requests.

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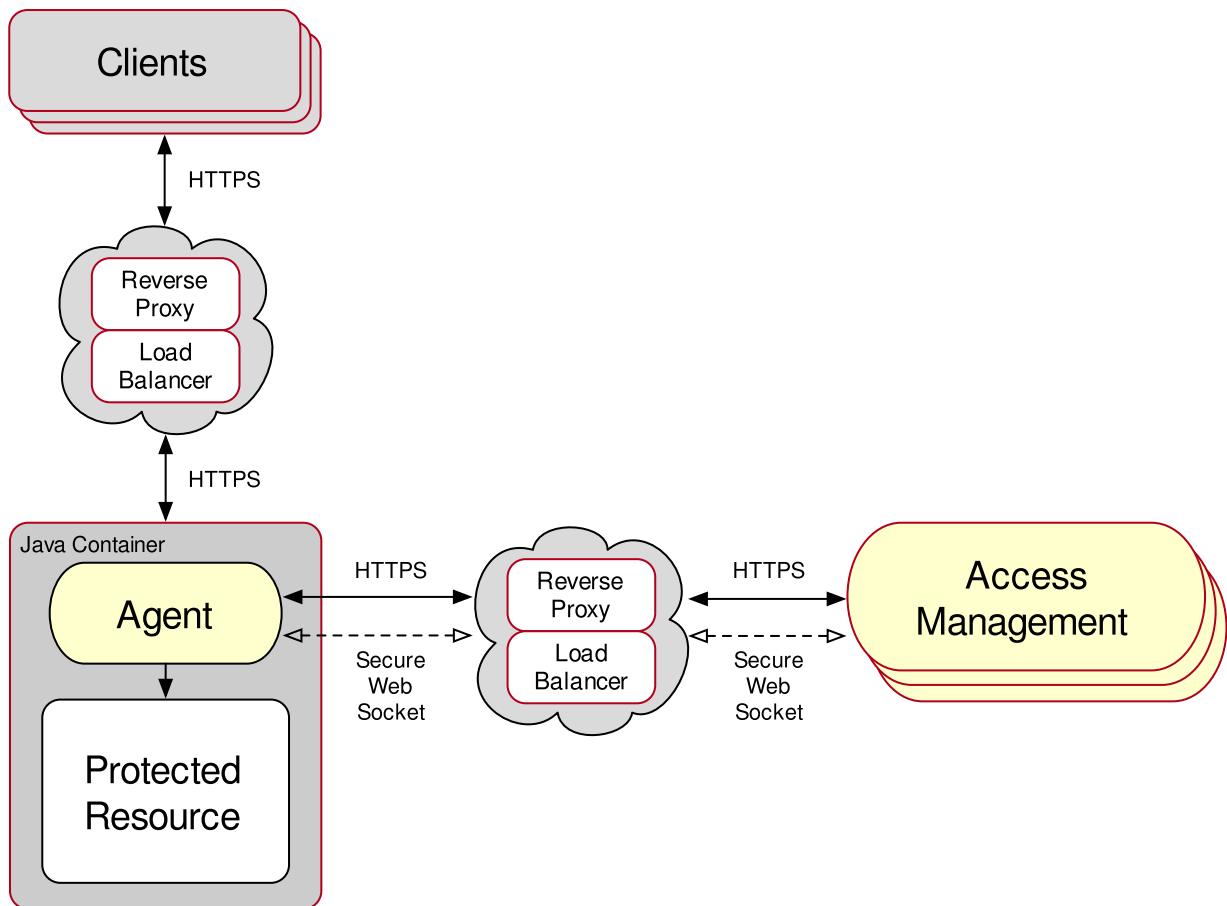
The reverse proxy and the load balancer can be the same entity. In complex environments, multiple load balancers and reverse proxies can be deployed in the network.

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.

Pre-authentication cookie signing

During installation, the agent requests the path to a file containing the cookie signing key, and then uses the key to configure the property `org.forgerock.agents.cookie.signing.value` in AgentKey.properties. If the path is empty, cookie signing is disabled.

The key must be at least 64 characters long. If it is shorter, the agent rejects it and leaves cookies unsigned. For security, use a key of at least 80 characters.

In deployments with multiple agent instances, use the same cookie signing key for each instance. Share a cookie signing key between agent instances as follows:

- If agent instances can share the signing key file, specify the same file for each agent installation.

- If agent instances are in remote configuration mode, set the property `org.forgerock.agents.cookie.signing.value` in the **Advanced** tab of the AM admin UI.

  When storing shared keys in AM secure communication between the agent and AM.

- Manually update the agent configuration, as defined in the following procedure:

  1. Install the first agent instance.

  2. Note the value of `org.forgerock.agents.cookie.signing.value` in the AgentKey.properties file.

  3. Install the next agent instance, and then replace the value of `org.forgerock.agents.cookie.signing.value` in AgentKey.properties with the value from the first instance.

  4. Restart the instance.

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, `amlbcookie`) 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 an Apache HTTP Server as a reverse proxy.

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 admin UI 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, `/am`.

     Leave the other fields empty.

     For more information, 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 TLS offloading

The load balancer or reverse proxy performs TLS offloading, terminating the TLS 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 proxie 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.
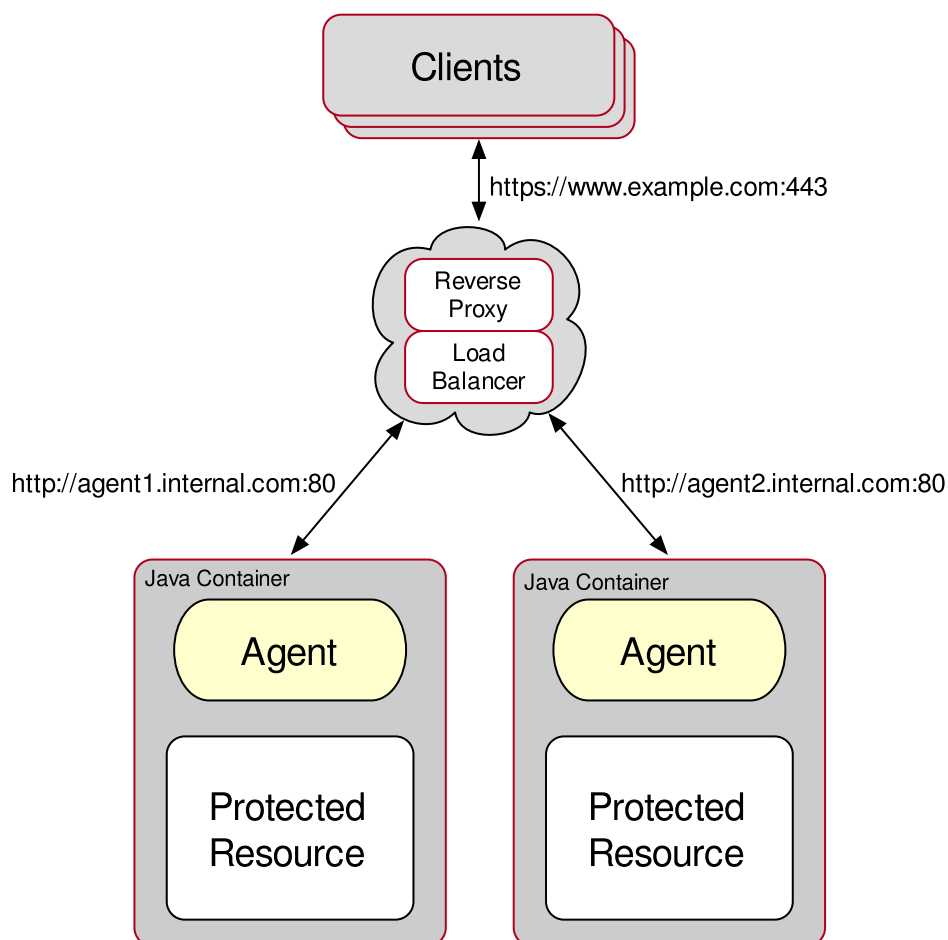
After TLS offloading, the host, port, and protocol of the request is changed to match the request received by the agent; it no longer matches the request from the client, as shown in the following data flow. The agent uses this URL for the redirect_url from the OAuth 2.0 flow, which causes the request to fail.
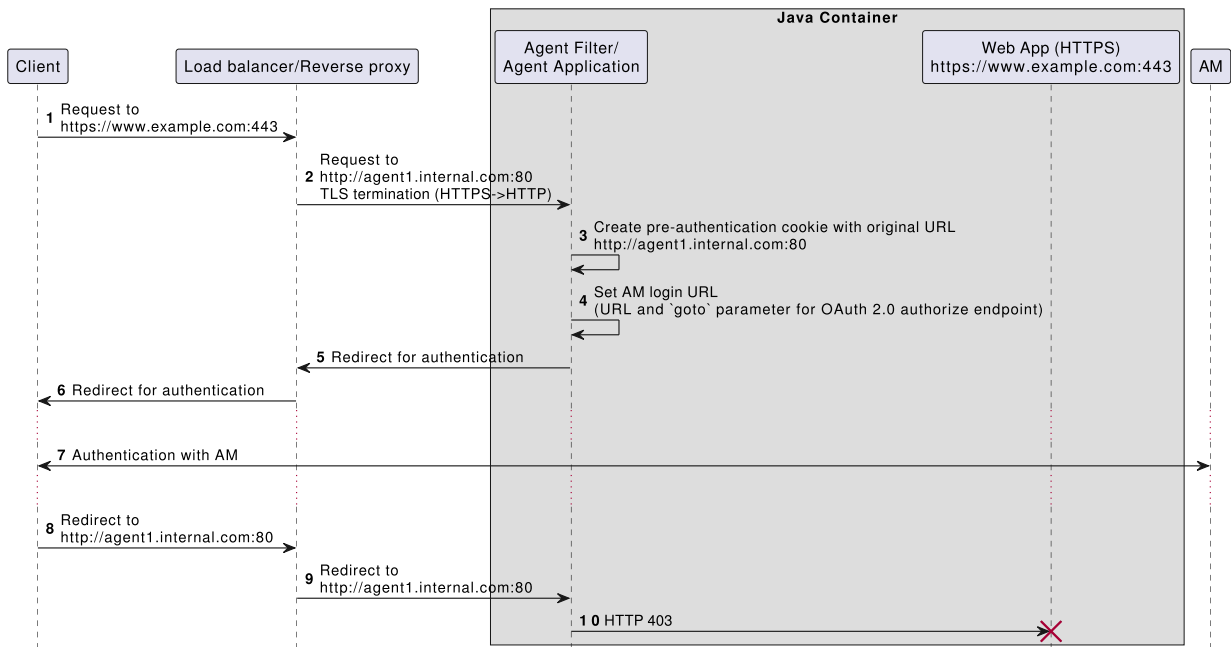
In the following flow, the agent overrides the host, port, and protocol for subsequent redirects:
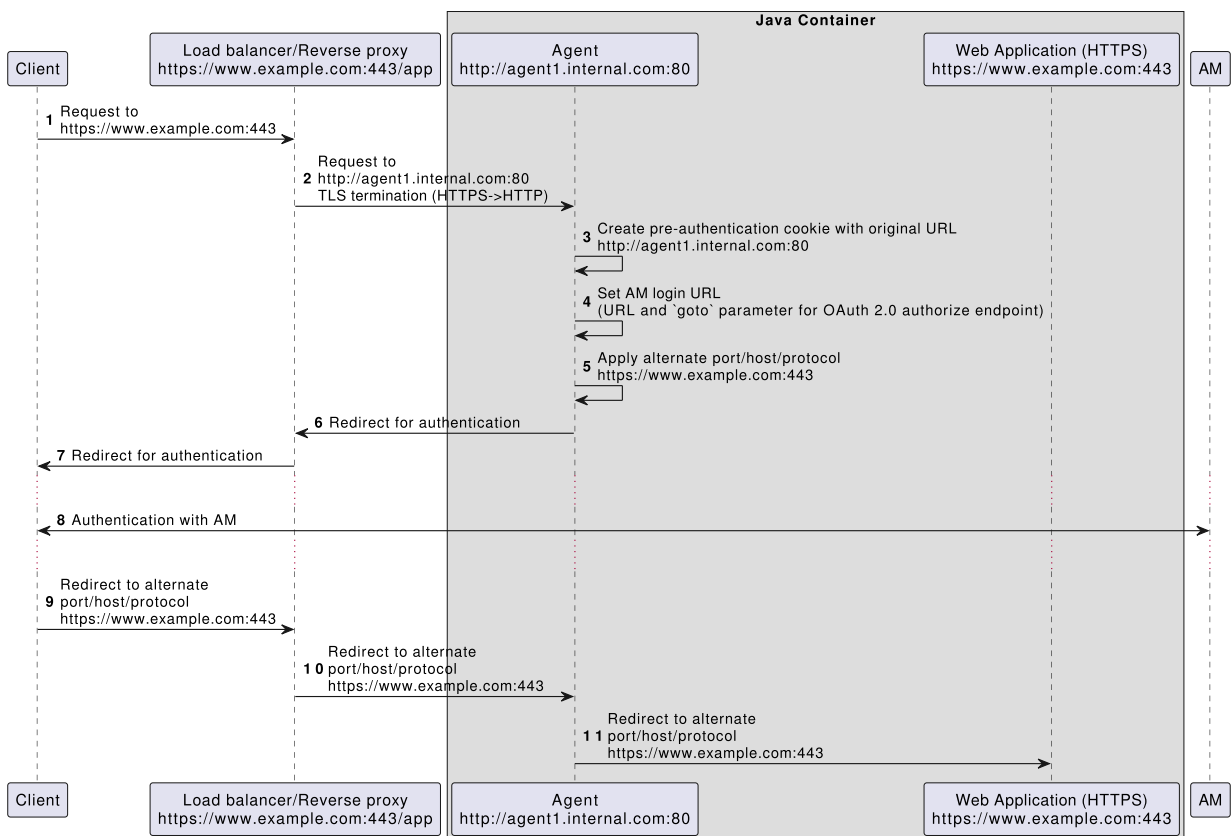
For this scenario, configure the agent as described in <u>To Override Protocol, Host, and Port</u>.

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:
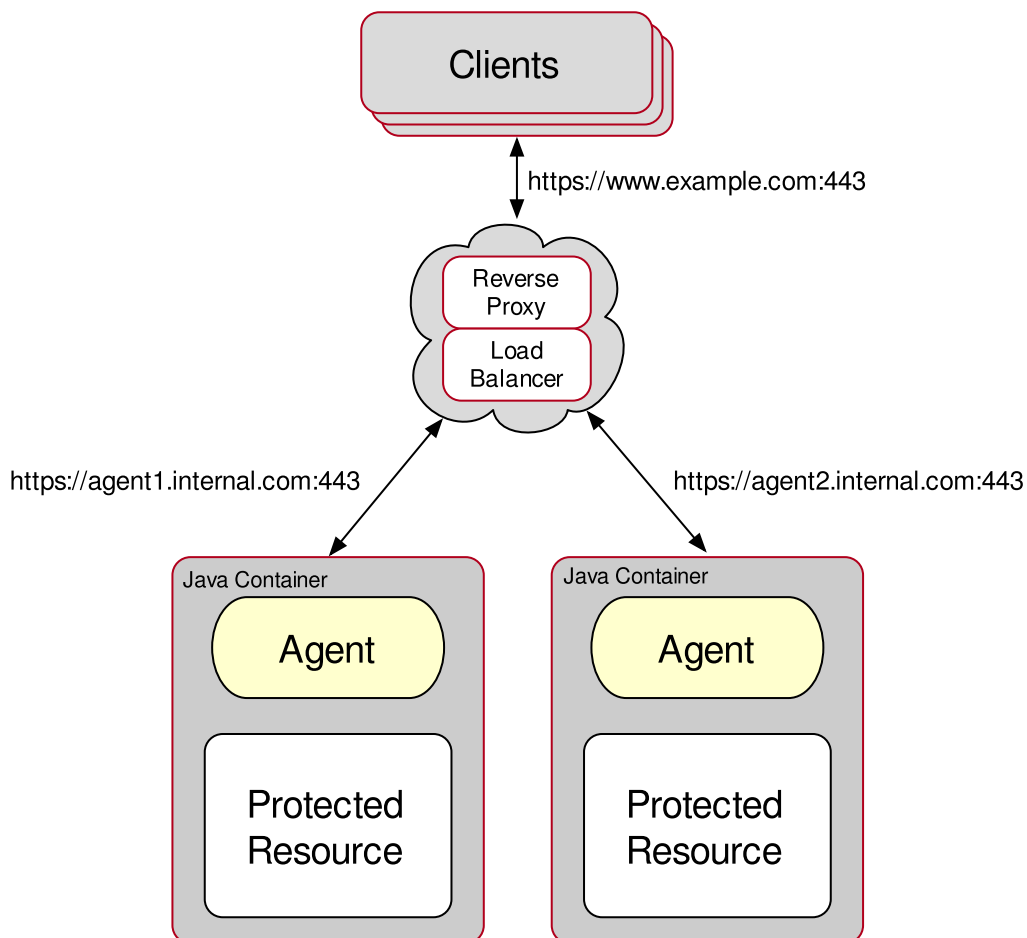
For this scenario, configure the agent as described in <u>To Map the Agent Host Name to the Load Balancer or Reverse Proxy Host Name</u>.

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 <u>remote configuration mode</u>, but the steps mention properties for agents in <u>local configuration mode</u>.

IMPORTANT

Agent configuration for TLS 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 admin UI 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 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 admin UI 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 CDSSO**.

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 admin UI 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 admin UI 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.

Configure an Apache HTTP Server as a reverse proxy

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.
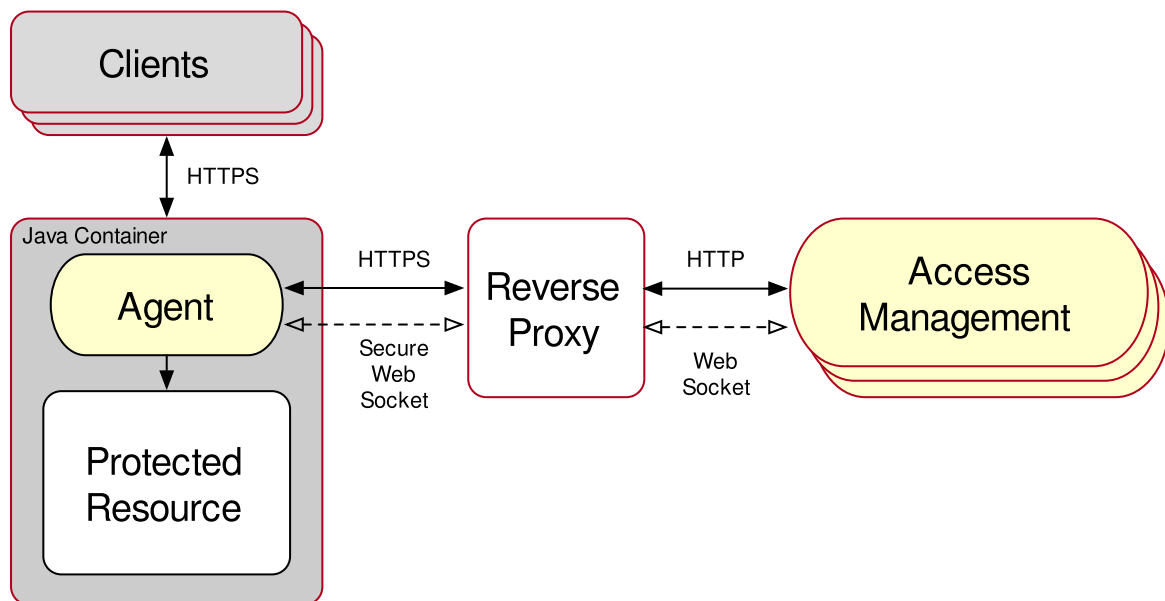

Figure 6. Reverse Proxy Configured Between the Agent and AM

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 "/am/notifications"
```

```
"ws://am.example.com:8080/am/notifications"
Upgrade=websocket ②
ProxyPass "/am" "http://am.example.com:8080/am" ③
ProxyPassReverseCookieDomain "am.internal.example.com"
"proxy.example.com" ④
ProxyPassReverse "/am" "http://am.example.com:8080/am" 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

```java
/*
 * Copyright 2019-2023 ForgeRock AS. All Rights Reserved
 *
 * Use of this code requires a commercial software license
with ForgeRock AS.
 * or with one of its affiliates. All use shall be exclusively
subject
 * to such license between the licensee and ForgeRock AS.
 */
package com.sun.identity.agents.custom;

import static org.forgerock.agents.debug.AgentDebug.logTrace;

import javax.servlet.http.HttpServletRequest;

import org.forgerock.agents.util.Utils;

import com.sun.identity.agents.arch.AgentConfiguration;
import com.sun.identity.agents.arch.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";
    }
```

```java
    @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

```java
/*
 * Copyright 2019-2023 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;
```

```java
import org.forgerock.agents.util.Utils;

import com.sun.identity.agents.arch.AgentConfiguration;
import com.sun.identity.agents.arch.AgentException;
import com.sun.identity.agents.arch.Manager;
import com.sun.identity.agents.filter.AmFilterMode;
import com.sun.identity.agents.filter.AmFilterRequestContext;
import com.sun.identity.agents.filter.AmFilterResult;
import com.sun.identity.agents.filter.AmFilterTaskHandler;
import com.sun.identity.agents.filter.IBaseAuthnContext;

/**
 * This is an example of a custom self-redirect task handler.
It is essentially the same as the inbound task
 * handler.
 */
@SuppressWarnings("unused")
public class CustomSelfRedirectTaskHandler extends
AmFilterTaskHandler {

    public CustomSelfRedirectTaskHandler(Manager manager) {
        super(manager);
    }

    @Override
    public void initialize(IBaseAuthnContext context) throws
AgentException {
        super.initialize(context);
    }

    @Override
    public boolean isActive() {
        return true;
    }

    @Override
    public String getHandlerName() {
        return "Custom self redirect task handler";
    }

    @Override
    public AmFilterResult process(AmFilterRequestContext
context) {
```

```
        String applicationName =
Utils.getApplicationName(context);
        AmFilterMode amFilterMode =
AgentConfiguration.getTheFilterMode(applicationName);
        HttpServletRequest request =
context.getHttpServletRequest();

        logTrace("Hello from {}, application name {}, filter
mode {}, {} {}",
                getHandlerName(), applicationName,
amFilterMode,
                request.getMethod(), request.getRequestURI());

        // return null to continue to the other task handlers
(until one returns a non null value)
        // return AmFilterResultStatus.STATUS_CONTINUE to
grant access (continue to the next filter after the agent)
        // return AmFilterResultStatus.STATUS_REDIRECT to
redirect somewhere else
        // return AmFilterResultStatus.STATUS_FORBIDDEN to
deny access
        // return AmFilterResultStatus.STATUS_SERVE_DATA to
serve up data to the browser
        // return AmFilterResultStatus.STATUS_SERVER_ERROR to
abort the request with a 500 server error
        //
        return null;
    }
}
```

Example custom inbound task handler

```
/*
 * Copyright 2019-2023 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;
```

```java
import static org.forgerock.agents.debug.AgentDebug.logTrace;

import javax.servlet.http.HttpServletRequest;

import org.forgerock.agents.util.Utils;

import com.sun.identity.agents.arch.AgentConfiguration;
import com.sun.identity.agents.arch.AgentException;
import com.sun.identity.agents.arch.Manager;
import com.sun.identity.agents.filter.AmFilterMode;
import com.sun.identity.agents.filter.AmFilterRequestContext;
import com.sun.identity.agents.filter.AmFilterResult;
import com.sun.identity.agents.filter.AmFilterTaskHandler;
import com.sun.identity.agents.filter.IBaseAuthnContext;

/**
 * This is an example of a custom inbound task handler
 */
@SuppressWarnings("unused")
public class CustomInboundTaskHandler extends
AmFilterTaskHandler {

    public CustomInboundTaskHandler(Manager manager) {
        super(manager);
    }

    @Override
    public void initialize(IBaseAuthnContext context) throws
AgentException {
        super.initialize(context);
    }

    @Override
    public boolean isActive() {
        return true;
    }

    @Override
    public String getHandlerName() {
        return "Custom inbound task handler";
    }

    @Override
    public AmFilterResult process(AmFilterRequestContext
```

```
context) {

        String applicationName =
Utils.getApplicationName(context);
        AmFilterMode amFilterMode =
AgentConfiguration.getTheFilterMode(applicationName);
        HttpServletRequest request =
context.getHttpServletRequest();

        logTrace("Hello from {}, application name {}, filter
mode {}, {} {}",
                getHandlerName(), applicationName,
amFilterMode,
                request.getMethod(), request.getRequestURI());

        // return null to continue to the other task handlers
(until one returns a non null value)
        // return AmFilterResultStatus.STATUS_CONTINUE to
grant access (continue to the next filter after the agent)
        // return AmFilterResultStatus.STATUS_REDIRECT to
redirect somewhere else
        // return AmFilterResultStatus.STATUS_FORBIDDEN to
deny access
        // return AmFilterResultStatus.STATUS_SERVE_DATA to
serve up data to the browser
        // return AmFilterResultStatus.STATUS_SERVER_ERROR to
abort the request with a 500 server error
        //
        return null;
    }
}
```

Example of how to override the ServiceResolver class

```
/*
 * Copyright 2019-2023 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.
```

```java
 */
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
```

```java
    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<>();
    }
}
```

Troubleshooting

ForgeRock provides support services, professional services, training through ForgeRock University, and partner services to help you set up and maintain your deployments. For help, refer to 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

▼ Detect the path of a resource loaded by classloader

Question: Why does the agent fails to initialize, with an error like this:

```
Servlet failed with an Exception: java.lang.NoSuchMethodError:
com.sun.identity.shared.configuration.SystemPropertiesManager.ge
tAsInt(Ljava/lang/String;I)I
```

Answer: An out-of-date version of the SystemPropertiesManager class is deployed in another jar, that did not have the `getAsInt` function, and the classloader chose the outdated class instead of the agent's class.

Follow these steps to help locate jars containing outdated classes, by showing where these classes are loaded from:

1. Add the property `-Ddisplay.classpath.mode.enabled=true` to the JVM properties in the container where the agent runs, and restart the container.

   ```
   JAVA_OPTS="$JAVA_OPTS -
   Ddisplay.classpath.mode.enabled=true"
   ```

2. Access the URL of a protected resource, using the class name as a query parameter. The following example requests the path of the SystemPropertiesManager package:

   ```
   https://www.example.com/myapp/index.html?
   com.sun.identity.shared.configuration.SystemPropertiesMan
   ager
   ```

   If the agent finds the package, it displays the path. Otherwise, it displays an error.

3. After troubleshooting, remove `Ddisplay.classpath.mode.enabled=true` from the JVM properties. While it is present, the agent returns the class path but performs no other function.

▼ Cannot install over HTTPS

Question:

What should I do if I get an error like this when installing an agent with an HTTPS connection to AM:

```
AM server URL: https://am.example.com:8443/am

WARNING: Unable to connect to AM server URL....

If the AM server is TLS-enabled and the root CA certificate for
the AM
server certificate has been not imported into the 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 the AM server URL.
```

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.

For information about export and import of self-signed certificates, see Configuring AM's container for HTTPS of AM's *Installation guide*.

▼ 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 <u>Alternative Agent Host Name</u> to access the agent through different virtual host names. For example, `http://agent.example.com/` and `http://internal.example.com/`.

- Configuring <u>Alternative Agent Port Number</u> to access the agent through different ports. For example, `http://agent.example.com/` and `http://agent.example.com:8080/`.

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 <u>Agent Filter Mode Map</u>.

Agent profile

A set of configuration properties that define the behavior of the agent.

Agent profile realm

The realm in which the agent profile is stored. See also <u>agent profile</u>.

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.

Custom login redirect

A mode to use SSO tokens or OpenID Connect (OIDC) ID tokens as session tokens, and redirect login to any endpoint. For more information, see Login redirect.

Default login redirect

A mode to use OpenID Connect (OIDC) ID tokens as session tokens, and redirect login to the `/oauth2/authorize` endpoint in the AM instance specified during installation. For more information, see Login redirect.

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.

Login redirect

Java Agent manages login redirect in the following ways: default login redirect and custom login redirect.

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 evaluation realm

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.

User realm

> The realm in which a user is authenticated.

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.