# User Guide

ON THIS PAGE

# User Guide

This guide describes how to use ForgeRock Access Management Web 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 Web Agent

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

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

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

  For information about how to enforce user authentication only, see Web Agent Single Sign-on (SSO) Only Mode.

This chapter covers how Web Agent works and how it protects applications.

## Agent Components

Web Agent includes the following main components:

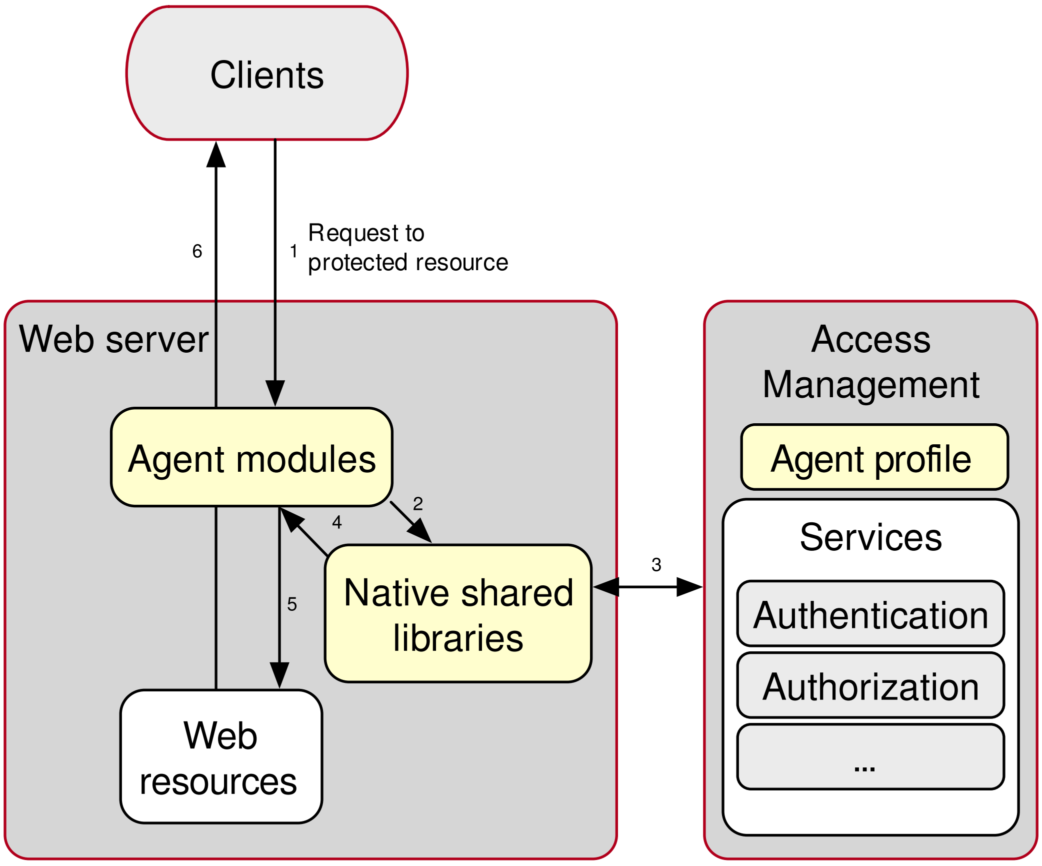
***Agent Modules***
Intercepts and processes inbound requests to protected resources.

***Native Shared Libraries***
Enables agents to interact with AM.

***Agent Profile***

The agent profile is not strictly part of Web Agent, but plays an important part in the agent operation. It contains a set of configuration properties that define the agent behavior.

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



*Configuration Location*

Web Agent configuration properties determine the behavior of the agent. AM stores configuration properties either centrally or locally:

*Centralized configuration*

AM stores the agent properties in the AM configuration store. Storing the agent configuration centrally allows you to configure your agents using the AM console, the `ssoadm` command, and the REST API.

To access the centralized web agent configuration, on the AM console go to Realms > Realm Name > Applications > Agents > Web > Agent Name.

Configure properties that are not present in the AM console as *custom properties*, on the Advanced tab of the console. For a list of property names, see the Properties Reference.

When properties and value pairs are defined as custom properties, they take precedence for that property. Therefore, to prevent configuration errors, do not configure a property as a custom property if it has a UI counterpart.

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

### *Local configuration*

The Web Agent installer creates the file `/web_agents/agent_type/instances/agent_nnn/config/agent.conf` to store the agent configuration properties.

The installer populates the file with enough information to make the agent start. To manage the configuration, edit the file to add properties, remove properties, and change values. You cannot update this file using the AM console, the **`ssoadm`** command, or the REST API.

The `agent.conf` file must contain at least the following properties:

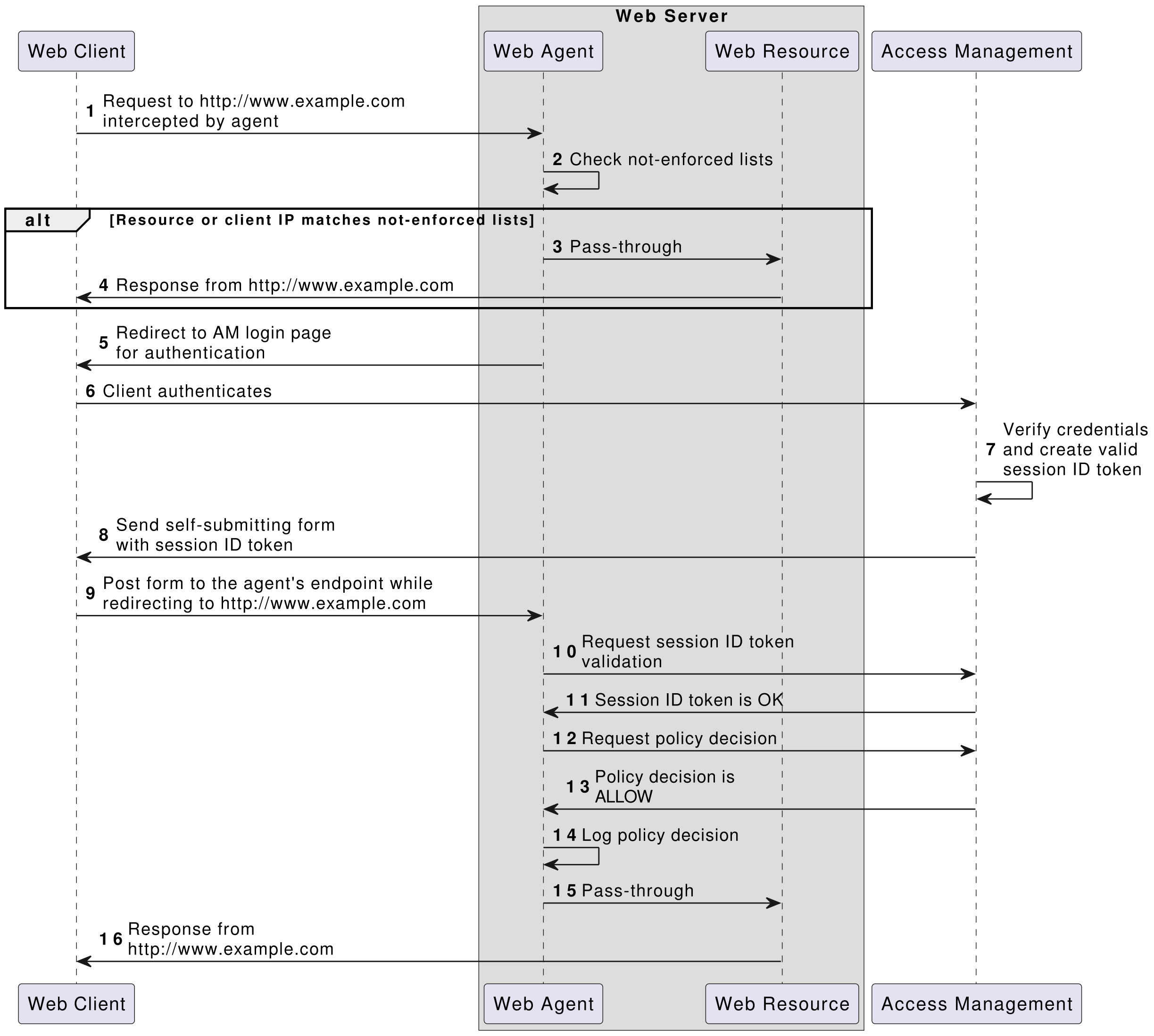
```
### Bootstrap properties
com.sun.identity.agents.config.organization.name = /
com.sun.identity.agents.config.username = ApacheAgentProfile
com.sun.identity.agents.config.password = o7OuvnaDnQ==
com.sun.identity.agents.config.key = OGM...0Yg=
com.sun.identity.agents.config.naming.url =
https://openam.example.com:8443/openam

### Configuration properties
com.sun.identity.agents.config.repository.location = local
org.forgerock.openam.agents.config.jwt.name = am-auth-jwt
com.sun.identity.agents.config.cdsso.redirect.uri = agent/cdsso-
oauth2
org.forgerock.openam.agents.config.policy.evaluation.application
= iPlanetAMWebAgentService
org.forgerock.openam.agents.config.policy.evaluation.realm = /
com.sun.identity.agents.config.polling.interval = 60
com.sun.identity.agents.config.sso.cache.polling.interval = 3
com.sun.identity.agents.config.policy.cache.polling.interval = 3
com.sun.identity.agents.config.cookie.name = iPlanetDirectoryPro
com.sun.identity.agents.config.debug.file.size = 10000000
com.sun.identity.agents.config.local.logfile =
/web_agents/agent_type/instances/agent_1/logs/debug/debug.log
com.sun.identity.agents.config.local.audit.logfile =
/web_agents/agent_type/instances/agent_1/logs/audit/audit.log
com.sun.identity.agents.config.debug.level = Error
```

The properties are provided with an example value. For information about each of these properties, see the Properties Reference.

### *Request Process Flow*

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

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


Web Server
Web Client
Web Agent
Web Resource
Access Management
1 Request to http://www.example.com intercepted by agent
2 Check not-enforced lists
alt [Resource or client IP matches not-enforced lists]
3 Pass-through
4 Response from http://www.example.com
5 Redirect to AM login page for authentication
6 Client authenticates
7 Verify credentials and create valid session ID token
8 Send self-submitting form with session ID token
9 Post form to the agent's endpoint while redirecting to http://www.example.com
10 Request session ID token validation
11 Session ID token is OK
12 Request policy decision
13 Policy decision is ALLOW
14 Log policy decision
15 Pass-through
16 Response from http://www.example.com

1. An unauthenticated client attempts to access a resource, and the agent intercepts the inbound request.
2. The agent evaluates whether the requested resource or the client IP address matches a not-enforced rule..
3. *Alternate Flow*. The requested resource or the client IP address matches a not-enforced rule. The agent allows access to the resource.

4. *Alternate Flow*. The client receives a response from `www.example.com`. The flow ends.
5. The requested resource or the client IP address does not match a not-enforced rule. The agent redirects the client to log in to AM.
6. The client authenticates to AM.

   During client authentication, and to protect against reply attacks, the agent issues a pre-authentication cookie, named `agent-authn-tx`. The agent uses the cookie to track the authentication request to AM, and deletes it immediately after authentication.

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

   The pre-authentication cookie expires after 5 minutes, or after the time specified in Profile Attributes Cookie Maxage.

   If POST data preservation is enabled, the request expires after the time specified in POST Data Entries Cache Period, which is by default 10 minutes. In this case, consider increasing Profile Attributes Cookie Maxage to at least 10 minutes.
7. AM's Authentication Service verifies the client credentials and creates a valid OpenID Connect (OIDC) JSON Web Token (JWT) with session information.
8. AM sends the client a self-submitting form with the OIDC JWT.
9. The client posts the self-submitting form to the agent endpoint, and the Web Agent consumes it.
10. The agent contacts AM to validate the session contained in the ID token.
11. AM validates the session.
12. The agent contacts AM's Policy Service, requesting a decision about whether the client is authorized to access the resource.
13. AM's policy service returns `ALLOW`.
14. The agent writes the policy decision to the audit log.
15. 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.
16. The client accesses the resource.

## Features

### Multiple Sites and Virtual Hosts

Web Agent instances can be configured to operate with multiple websites in IIS, and with multiple virtual hosts in Apache.

Each configuration instance is independent and has its own configuration file, debug logs, and audit logs. Each instance can connect to a different AM realm, or even different AM servers.

For more information, see Installing Apache Web Agents on a Virtual Host and Installing the IIS Web Agent.

## SSO-Only Mode

Web Agent intercepts all inbound client requests to access a protected resource and processes the request based on the Enable SSO Only Mode property.

The configuration setting determines the mode of operation that should be carried out on the intercepted inbound request, as follows:

- When `true`, the agent manages user authentication only. The filter invokes the AM Authentication Service to verify the identity of the user. If the identity is verified, the user is issued a session token through AM's Session Service.
- When `false`, which is the default, the agent also manages user authorization, by using the policy engine in AM.

## Not-Enforced Rules

Not-enforced rules are configured in the `Application` tab of the AM console. Rules that apply an HTTP method filter are configured as Custom Properties, in the `Advanced` tab.

When a not-enforced rule applies to a request, the request is processed without authentication. Configure lists of the following not-enforced rules:

***Not-enforced URL rules***
Allow access to resources, such as images, stylesheets, or the HTML pages that provide the public front end of your site. See Not-Enforced URL List.

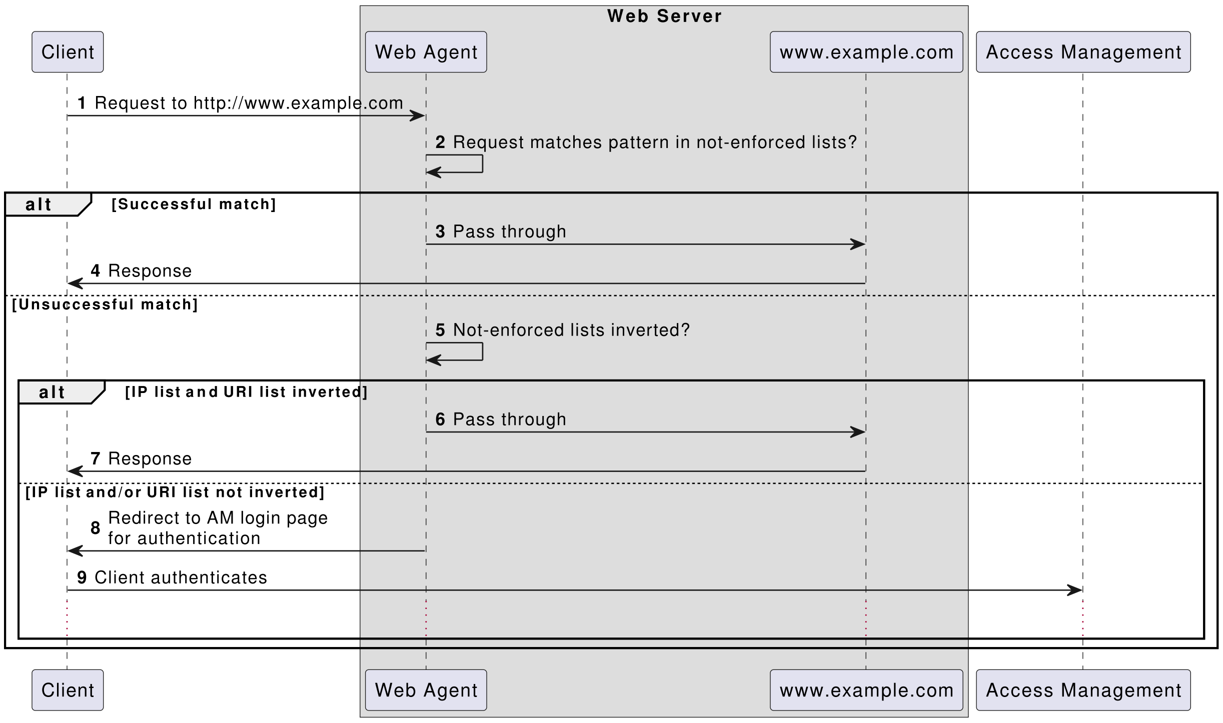
***Not-enforced IP rules***
Allow access to your site from an administrative IP address, an internal network range, or a search engine. See Not-Enforced IP List.

***Compound not-enforced rules***
Allow access to listed IP addresses when requesting the listed URLs. See Not-Enforced URL from IP Processing List.

When there are multiple lists of rules, the Web Agent evaluates them in this order:

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

The following image shows the data flow when Web Agent evaluates not-enforced rules for a request:


Web Server
Client
Web Agent
www.example.com
Access Management
1 Request to http://www.example.com
2 Request matches pattern in not-enforced lists?
alt
[Successful match]
3 Pass through
4 Response
[Unsuccessful match]
5 Not-enforced lists inverted?
alt
[IP list and URI list inverted]
6 Pass through
7 Response
[IP list and/or URI list not inverted]
8 Redirect to AM login page for authentication
9 Client authenticates

1. A client requests a resource.

2. The Web Agent checks whether the request matches a rule in a not-enforced list.

The Web 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.

3-4. The Web Agent passes the request without requiring the client to authenticate.

5-9. If the request doesn't match a rule in a not-enforced list, the Web Agent checks whether it matches a not-enforced URL rule that is inverted. If so, the Web Agent passes the request without requiring the client to authenticate.

## *Conventions for Not-Enforced Rules*

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

### *Invert All Rules*

Invert all rules in a not-enforced URL list by setting Invert Not-Enforced URLs to `true`.

Consider the following points when you invert all rules:

- If Not-Enforced URL List is empty, all URLs are enforced.

- At least one URL must be enforced. To allow access to any URL without authentication, consider disabling the agent.

### *Wildcard for not-enforced IP rules*

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

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

```
com.sun.identity.agents.config.notenforced.ip[1]=192.168.10.*
```

For more information about using wildcards, see Specifying Resource Patterns with Wildcards.

### *Wildcards for not-enforced URL rules*

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

  The wildcard spans multiple levels. For example:

  ```
  com.sun.identity.agents.config.notenforced.url[1]=http://www.e
  xample.com:8080/images/*
  com.sun.identity.agents.config.notenforced.url[2]=http://www.e
  xample.com:8080/*.jsp?locale=*
  ```

  Multiple forward slashes do not match a single forward slash. Therefore, `*` matches `mult/iple/dirs`, but `mult/*/dirs` does not match `mult/dirs`.
- Add `-*-` to match all characters in a rule, except the forward slash `/` and the question mark `?` character. The wildcard cannot be escaped. Because this wildcard does not match the `/` character, it does not span multiple levels in a URL. For example:

  ```
  com.sun.identity.agents.config.notenforced.url[1]=http://www.e
  xample.com:8080/css/-*-
  ```
- `*` and `-*-` wildcards cannot be used in the same rule, but can be used in different rules in the same list. For example:

  ```
  com.sun.identity.agents.config.notenforced.url[1]=http://www.e
  xample.com:8080/css/-*-
  ```

```
com.sun.identity.agents.config.notenforced.url[2]=http://www.e
xample.com:8080/images/*
```

- Multiple wildcards in the query parameter section of a not-enforced URL rule match the parameters in any order that they appear in a resource URL. For example, the following not-enforced URL rule applies to any resource URL that contains a `member_level` and `location` query parameter, in any order:

  ```
  com.sun.identity.agents.config.notenforced.url[1]=http://www.e
  xample.com:8080/customers/*?*member_level=*&location=*
  ```

  In following example, the requests would be not-enforced:

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

  If the parameters are not present in the request, the agent evaluates the resource URL against policies in AM, as usual.

- Trailing forward slashes are not recognized as part of a resource name. Therefore, `/images//` and `/images` are equivalent. For more information about using wildcards, see Specifying Resource Patterns with Wildcards.

*Regular expressions*

Set Regular Expressions for Not-Enforced URLs to `true`, and consider the following points for using regular expressions in not-enforced rules:

- 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:

  ```
  com.sun.identity.agents.config.notenforced.ip[1]=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*

Rules that apply an HTTP method filter are configured as Custom Properties in AM.

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 OPTIONS requests to your scripts, but does require authentication for other HTTP methods:

```
com.sun.identity.agents.config.notenforced.url[OPTIONS,1]=http://
www.example.com:8080/scripts/*
```

To specify a list of methods, add multiple rules:

```
com.sun.identity.agents.config.notenforced.url[OPTIONS,1]=http://
www.example.com:8080/scripts/*
com.sun.identity.agents.config.notenforced.url[TRACE,2]=http://ww
w.example.com:8080/scripts/*
```

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

```
com.sun.identity.agents.config.notenforced.ip[!POST,1]=192.168.1.
0/24
```

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

### *Compound rules*

Configure compound rules in Not-Enforced URL from IP Processing List.

In the following example, the agent does not enforce HTTP requests from the IP addresses `192.6.8.0/24` to any file in `/public`, or any files or directories that start with the string `login` in the directory `/free_access URI`:

```
org.forgerock.agents.config.notenforced.ipurl[1]=192.6.8.0/24|htt
p://www.example.com:8080/public/* /free_access/login*
```

### *Encoding non-ASCII characters in rules*

Percent-encode resources that use non-ASCII characters.

For example, to match resources to the URI `http://www.example.com/forstå`, specify the following percent-encoded rule:

```
/forst%C3%A5/*
```

## Attribute Fetch Modes

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

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

For example, you can have a web page that addresses the user by name retrieved from the user profile, for example "Welcome Your-Name!". AM populates part of the request (header, form data) with the CN from the user profile, and the website consumes and displays it.

## FQDN Checking

Web Agent requires that clients use valid URLs with FQDNs to access protected resources. If invalid URLs are referenced, policy evaluation can fail because the FQDN does not match the requested URL, leading to blocked access to the resource. Misconfigured URLs can also result in incorrect policy evaluation for subsequent access requests.

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

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

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

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

## Cookie Reset

Web Agent can reset cookies before redirecting the client to a login page, by issuing a Set-Cookie header to the client to reset the cookie values.

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

NOTE

To set and reset secure or HTTP Only cookies, in addition to the cookie reset properties, set the relevant cookie option, as follows:

- To reset secure cookies, enable the `com.sun.identity.agents.config.cookie.secure` property.
- To reset HTTP only cookies, enable the `com.sun.identity.cookie.httponly` property.

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

## Cross-Domain Single Sign-On

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

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

Web Agent works in CDSSO mode by default, regardless of the DNS domain of the AM servers and the DNS domain of the web agents.

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

## Continuous Security

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

When Web Agent requests a policy decision from AM, it can include the additional information in an *environment map*, a set of name/value pairs that describe the request IP and DNS name, along with other, optional, information. The additional information can then be included in the policy, for example, to allow only incoming requests that contain the `InternalNetwork`.

In AM, use server-side authorization scripts to access the environment map, and write scripted conditions based on cookies and headers in the request. For information about server-side authorization scripts, see Scripting a Policy Condition in AM's *Authorization Guide*.

### *Environment Maps With Customizable Keys*

In Web Agent, use the continuous security properties Continuous Security Cookie Map and Continuous Security Header Map to configure an environment map with the following parts:

***requestIp***

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

- If Client IP Address Header is configured, Web Agent extracts the IP address from the header.
- Otherwise, Web Agent uses the container's connection information to determine the client IP address.

This entry is always created in the map.

***requestDNSName***

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

- If Client Hostname Header is configured, Web Agent extracts the host name from the header.
- Otherwise, Web Agent uses the container's connection information to determine the client's host name.

This entry is always created in the map.

***Other variable names***

An array of cookie or header values. 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.openam.agents.config.continuous.security.cookies[
ssid]=mySsid
org.forgerock.openam.agents.config.continuous.security.headers[
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
```

## Login Redirect

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

Default login redirects use OpenID Connect ID tokens as session tokens, and the AM UI end user pages to log in users. Use default login redirect for all new deployments.

Custom login redirects support environments with custom login pages, that are upgrading from Web Agents 4.x. Custom login redirects let the agent use AM-specific SSO tokens as session tokens. Use conditional login redirects to use the request URL as a criteria for redirects to different AM instances or sites, or authentication realms.

Configure login redirect options as follows:

***Login Redirect Configuration Options***

| | Accept SSO Token | |
|---|---|---|
| | 0 | 1 |

|  |  |  |  |
|---|---|---|---|
| Enable Custom Login Mode | 0 | • For AM 6 or later.<br>• Default login redirect<br>• Do not accept SSO tokens as session cookies | • Default login redirect<br>• Accepts SSO tokens and ID tokens as session tokens during and after the login flow.<br>• SSO tokens are not converted to ID tokens. |
|  | 1 | • Same domain custom login redirect or cross domain custom login redirect<br>• Redirection to the originally requested resource<br>• SSO tokens converted to ID tokens | • Not supported |
|  | 2 | • Not supported | • Non-standard flow; this configuration has limitations, and is only for environments migrating from earlier versions of the agent.<br>• For AM 6 or later.<br>• Same domain custom login redirect<br>• Redirection with a `goto` query parameter to the originally requested resource<br>• Accept SSO tokens |

### *Default Login Redirect*

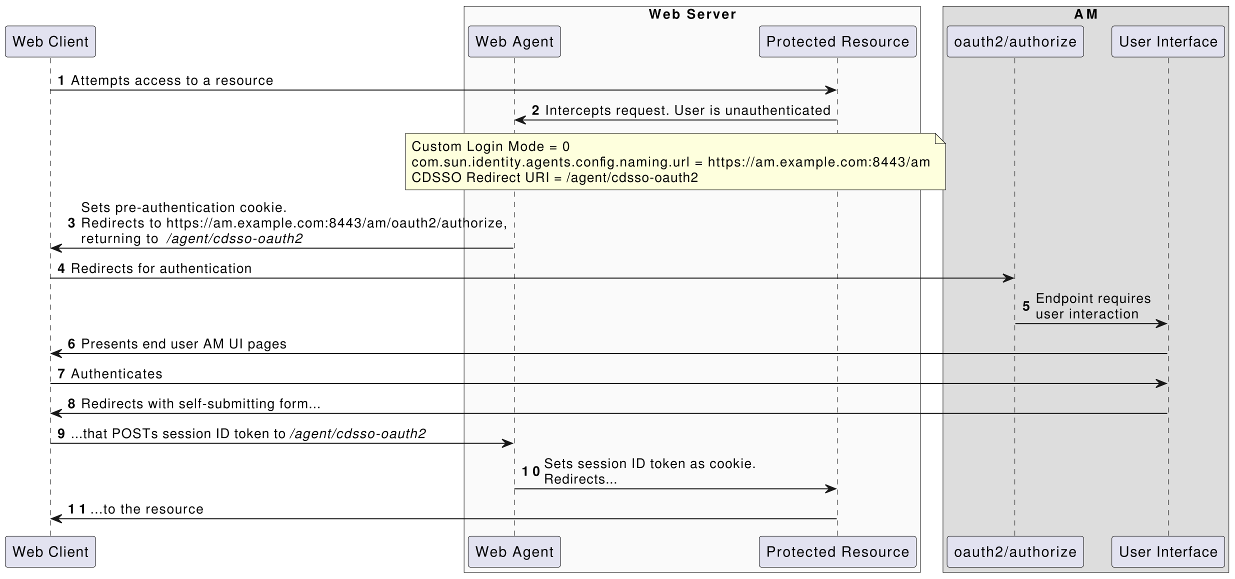
Unauthenticated users are redirected to, and must be able to access, the `/oauth2/authorize` endpoint. This endpoint invokes the AM console and other endpoints such as `oauth2/authorize`, `json/authenticate`, `json/sessions`, `json/serverinfo`, and `XUI/*`.

Set the following properties to configure default login redirects:

- Enable Custom Login Mode as `0`, to disable custom login mode.

- CDSSO Redirect URI to process authentication requests.

The following image shows the data flow for a default login redirect:


Web Server
AM
Web Client
Web Agent
Protected Resource
oauth2/authorize
User Interface
1 Attempts access to a resource
2 Intercepts request. User is unauthenticated
Custom Login Mode = 0
com.sun.identity.agents.config.naming.url = https://am.example.com:8443/am
CDSSO Redirect URI = /agent/cdsso-oauth2
3 Sets pre-authentication cookie. Redirects to https://am.example.com:8443/am/oauth2/authorize, returning to /agent/cdsso-oauth2
4 Redirects for authentication
5 Endpoint requires user interaction
6 Presents end user AM UI pages
7 Authenticates
8 Redirects with self-submitting form...
9 ...that POSTs session ID token to /agent/cdsso-oauth2
10 Sets session ID token as cookie. Redirects...
11 ...to the resource

## *Same Domain Custom Login Redirect*

The agent redirects unauthenticated users to a custom login page in the same domain, adding the `original_request_url` parameter to the redirect. The parameter records the requested URL, which can then be used by custom login application or page.

The custom login page posts an SSO token to `agent/custom-login-response`, with the realm as an optional parameter, and sets an SSO token in the same domain as the agent.

The agent attempts to validate the SSO token against the AM endpoints.

At the end of the login flow, the agent can do the final redirect to the protected resource, or to the originally requested resource, with a `goto` query parameter

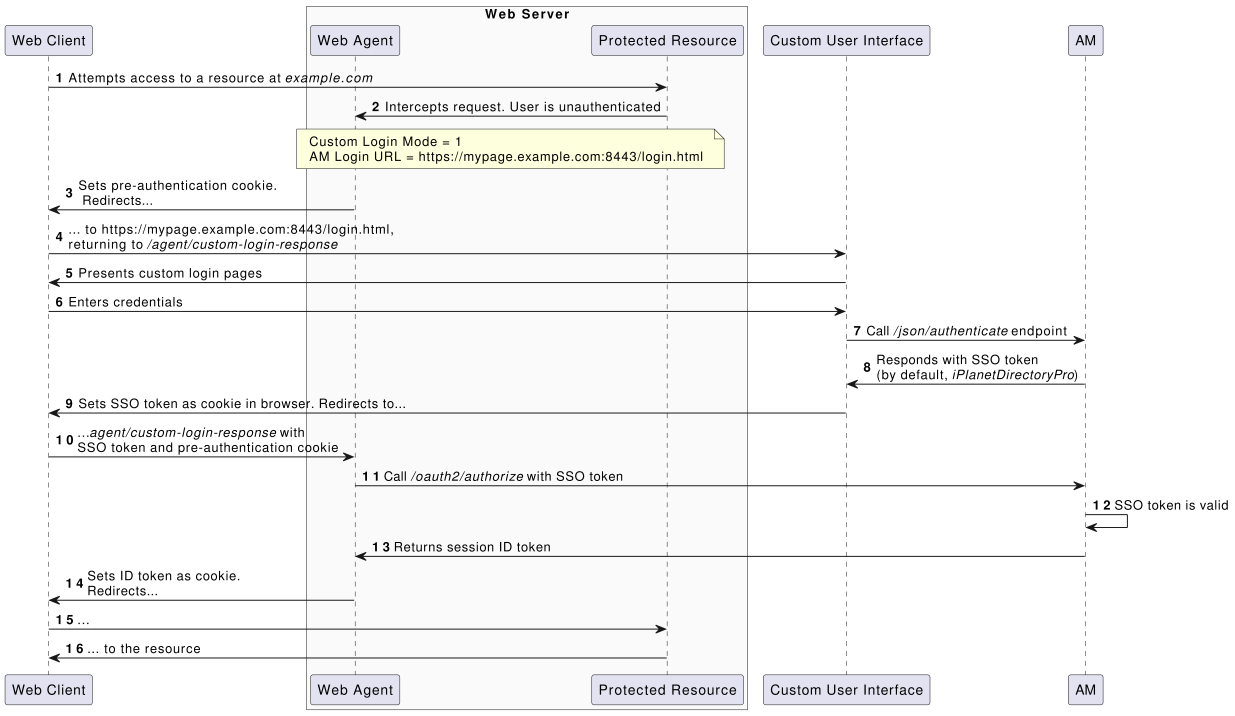
### *Same Domain Custom Login Redirect With Final Redirect to the Protected Resource*

Set the following properties:

- Enable Custom Login Mode as `1`, to use the OIDC-compliant custom login redirection mode
- AM Login URL, to the URL of the custom login page

If the custom login page is a part of the agent's protected application, add the custom login pages to the not enforced lists.

The following image shows the data flow for a custom login redirect when the custom login pages are in the same domain as the agent, and the agent redirects the the request to the originally requested resource.


Web Server
Web Client
Web Agent
Protected Resource
Custom User Interface
AM
1 Attempts access to a resource at *example.com*
2 Intercepts request. User is unauthenticated
Custom Login Mode = 1
AM Login URL = https://mypage.example.com:8443/login.html
3 Sets pre-authentication cookie. Redirects...
4 ... to https://mypage.example.com:8443/login.html, returning to */agent/custom-login-response*
5 Presents custom login pages
6 Enters credentials
7 Call */json/authenticate* endpoint
8 Responds with SSO token (by default, *iPlanetDirectoryPro*)
9 Sets SSO token as cookie in browser. Redirects to...
10 ...*agent/custom-login-response* with SSO token and pre-authentication cookie
11 Call */oauth2/authorize* with SSO token
12 SSO token is valid
13 Returns session ID token
14 Sets ID token as cookie. Redirects...
15 ...
16 ... to the resource
Web Client
Web Agent
Protected Resource
Custom User Interface
AM

### *Same Domain Custom Login Redirect With Final Redirect to the Originally Requested Resource (Migration Mode)*

In this scenario, the agent redirects the client with a `goto` query parameter to the originally requested resource.

This mode only operates on HTTP GET requests. POST requests are not supported.

CAUTION

This is not a standard flow, and this feature is evolving. Use it only when migrating from earlier versions of the agents. Contact ForgeRock if you suspect your environment has a similar use case.

Part of this flow happens outside the agent's control, and, therefore, the SSO token may expire or become invalid before the agent has a chance to validate it. In these cases, the user/client needs to authenticate again.

- The custom login pages obtain the SSO token from AM
- The agent **does not** track the user authentication using the pre-authentication cookie

### *Cross Domain Custom Login Redirect*

The agent redirects unauthenticated users to a custom login page in a different domain, including the `original-request-uri` parameter in the redirect. The parameter records the requested URL, which can then be used by custom login application or page.

The custom login page provides a `custom-login-response`, and sets an SSO token, which can be accessed only in that domain. Because the agent cannot access the cookie, it redirects to AM for the Default Login Redirection Mode.

Depending on your environment, the agent can contact AM to validate the cookie even if it cannot see it. In other cases, you need to configure an additional property.

If AM can validate the SSO token, it returns an ID token as part of the default redirection login flow.

Consider the following points:

- Ensure that the login pages **do not** set the SSO token cookie with the `SameSite=Strict` attribute.
- If AM cannot validate the SSO token (for example, because it cannot recognize the domain set for the cookie), it redirects the end user to authenticate again using the Default Login Redirection Mode.
- AM must be visible to the custom login pages, either because they both are in the same network/domain, or because you exposed the relevant AM endpoints using a proxy:

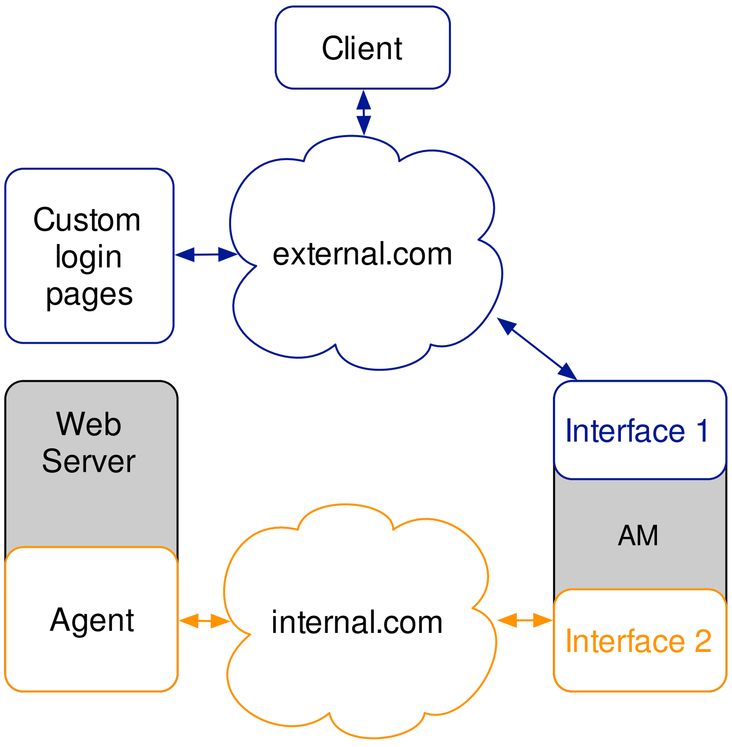
### *Cross Domain Custom Login Redirect on a Shared Network*

On a shared network, the server where AM is running has two interfaces: one connected to the internal network, where the agent is, and another connected to the external network, where the custom login pages are.

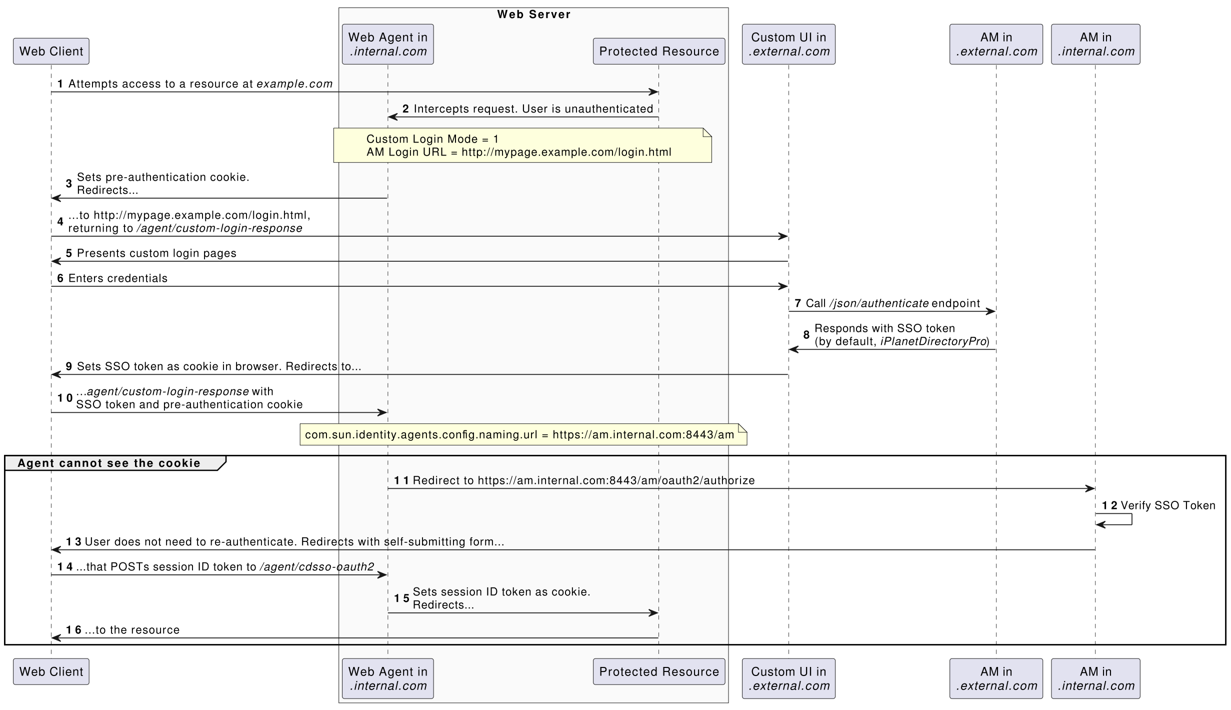
Use the following properties to configure this scenario:

- Enable Custom Login Mode
- AM Login URL

The following image illustrates the environment. The web server housing the protected resources can be connected to the external network in different ways; with two interfaces, or through a proxy. It is not important for the purposes of custom login, so it is not shown.



The following image shows the data flow:


Web Server
Web Client
Web Agent in .internal.com
Protected Resource
Custom UI in .external.com
AM in .external.com
AM in .internal.com
1 Attempts access to a resource at *example.com*
2 Intercepts request. User is unauthenticated
Custom Login Mode = 1
AM Login URL = http://mypage.example.com/login.html
3 Sets pre-authentication cookie. Redirects...
4 ...to http://mypage.example.com/login.html, returning to */agent/custom-login-response*
5 Presents custom login pages
6 Enters credentials
7 Call */json/authenticate* endpoint
8 Responds with SSO token (by default, *iPlanetDirectoryPro*)
9 Sets SSO token as cookie in browser. Redirects to...
1 0 ...*agent/custom-login-response* with SSO token and pre-authentication cookie
com.sun.identity.agents.config.naming.url = https://am.internal.com:8443/am
Agent cannot see the cookie
1 1 Redirect to https://am.internal.com:8443/am/oauth2/authorize
1 2 Verify SSO Token
1 3 User does not need to re-authenticate. Redirects with self-submitting form...
1 4 ...that POSTs session ID token to */agent/cdsso-oauth2*
1 5 Sets session ID token as cookie. Redirects...
1 6 ...to the resource
Web Client
Web Agent in .internal.com
Protected Resource
Custom UI in .external.com
AM in .external.com
AM in .internal.com

### *Cross Domain Custom Login Redirect With AM Behind a Proxy*

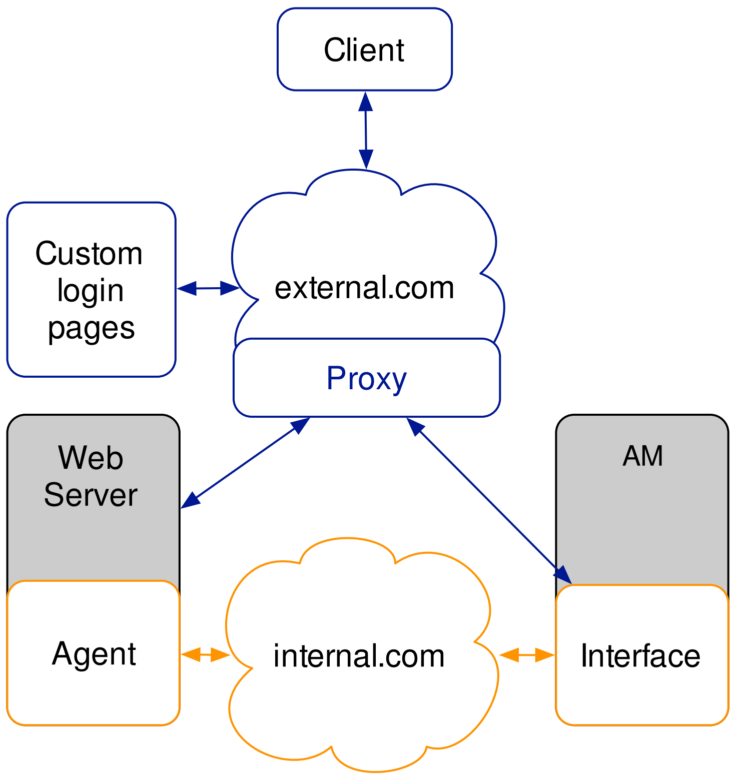
The server where AM is running has one interface to the internal network, where the agent is. A proxy hides AM from the external network, which forwards traffic to the `/oauth2/authorize` endpoint.

Use the following properties to configure this scenario:

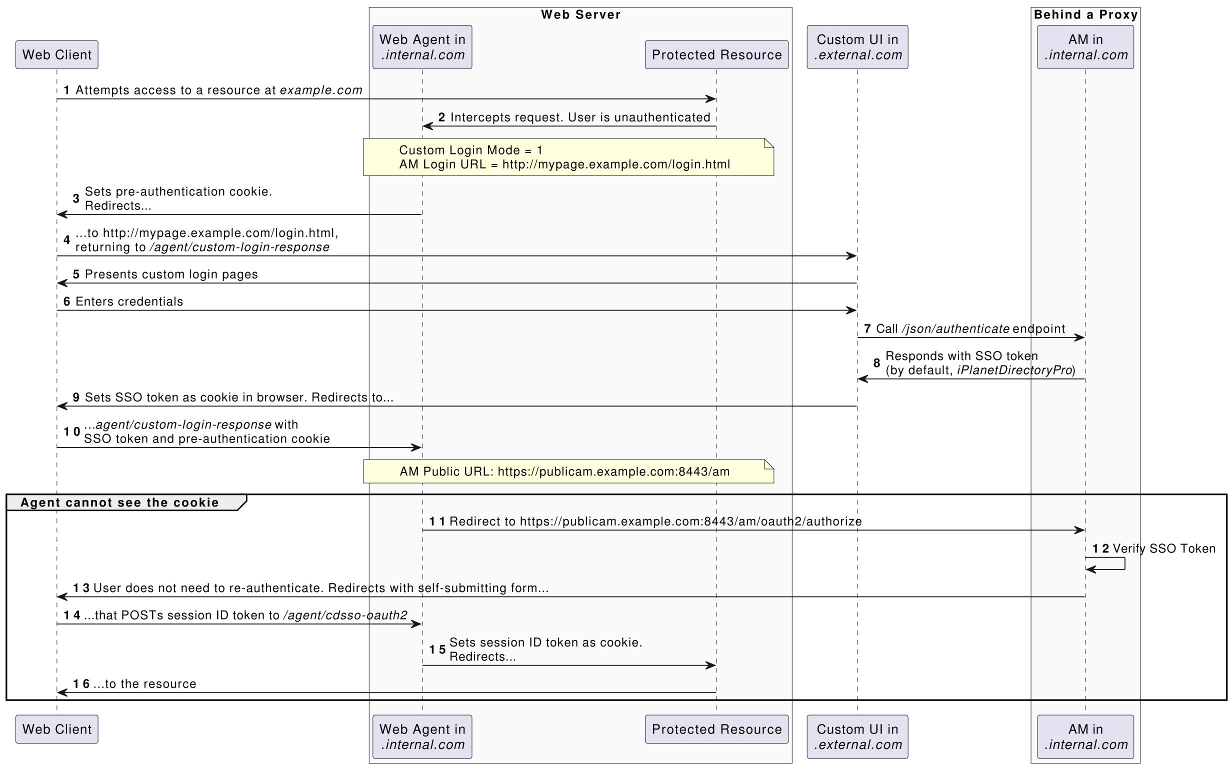
- Enable Custom Login Mode
- AM Login URL

- Public AM URL

The following image illustrates the environment. The web server where the protected resources are may be connected to the external network in different ways; with two interfaces, or through a proxy. It is not important for the purposes of custom login, so it is not shown in the following diagram:



The following image shows the data flow:



Web Server
Behind a Proxy
Web Client
Web Agent in .internal.com
Protected Resource
Custom UI in .external.com
AM in .internal.com
1 Attempts access to a resource at example.com
2 Intercepts request. User is unauthenticated
Custom Login Mode = 1
AM Login URL = http://mypage.example.com/login.html
3 Sets pre-authentication cookie. Redirects...
4 ...to http://mypage.example.com/login.html, returning to /agent/custom-login-response
5 Presents custom login pages
6 Enters credentials
7 Call /json/authenticate endpoint
8 Responds with SSO token (by default, iPlanetDirectoryPro)
9 Sets SSO token as cookie in browser. Redirects to...
10 ...agent/custom-login-response with SSO token and pre-authentication cookie
AM Public URL: https://publicam.example.com:8443/am
Agent cannot see the cookie
11 Redirect to https://publicam.example.com:8443/am/oauth2/authorize
12 Verify SSO Token
13 User does not need to re-authenticate. Redirects with self-submitting form...
14 ...that POSTs session ID token to /agent/cdsso-oauth2
15 Sets session ID token as cookie. Redirects...
16 ...to the resource
Web Client
Web Agent in .internal.com
Protected Resource
Custom UI in .external.com
AM in .internal.com

## *Conditional Login Redirect*

Use conditional redirects with default or custom login redirects to redirect the end user to different AM instances or sites, or to different custom pages, depending on the incoming request URL.

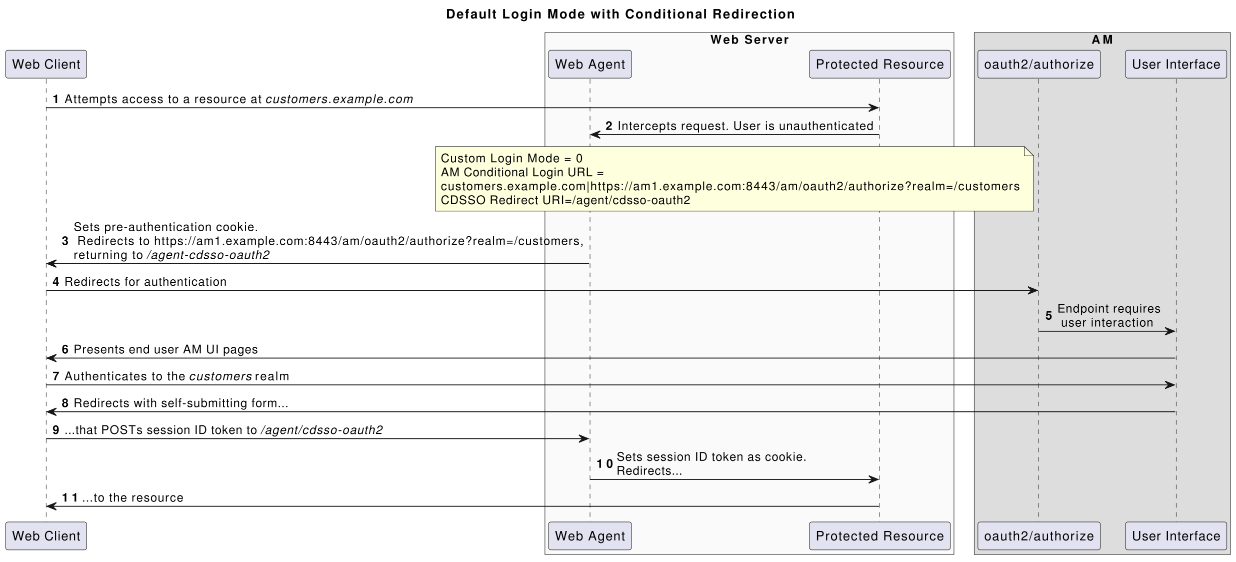
When the incoming request URL matches a specified domain name, configure the following properties to redirect to a specified URL:

- Enable Custom Login Mode
- AM Conditional Login URL
- CDSSO Redirect URI

When the incoming request URL matches a regular expression, configure the following properties to redirect to a specified URL:

- Enable Custom Login Mode
- AM Conditional Login URL
- Regular Expression Conditional Login Pattern
- CDSSO Redirect URI

The following image shows the data flow for a default login redirect flow when conditional redirection is configured:


Default Login Mode with Conditional Redirection
Web Server
AM
Web Client
Web Agent
Protected Resource
oauth2/authorize
User Interface
1 Attempts access to a resource at customers.example.com
2 Intercepts request. User is unauthenticated
Custom Login Mode = 0
AM Conditional Login URL =
customers.example.com|https://am1.example.com:8443/am/oauth2/authorize?realm=/customers
CDSSO Redirect URI=/agent/cdsso-oauth2
3 Sets pre-authentication cookie.
Redirects to https://am1.example.com:8443/am/oauth2/authorize?realm=/customers,
returning to /agent-cdsso-oauth2
4 Redirects for authentication
5 Endpoint requires user interaction
6 Presents end user AM UI pages
7 Authenticates to the customers realm
8 Redirects with self-submitting form...
9 ...that POSTs session ID token to /agent/cdsso-oauth2
10 Sets session ID token as cookie.
Redirects...
11 ...to the resource
Web Client
Web Agent
Protected Resource
oauth2/authorize
User Interface

## Logout Redirection

Web Agent can redirect users on logout to a specific AM page or to a custom logout page in your web server.

Configure logout redirect with the following properties:

- Logout URL List
- Agent Logout URL Regular Expression

- AM Logout URL
- Logout Redirect URL
- Enable Invalidate Logout Session
- Disable Logout Redirection
- Reset Cookies on Logout List

Logout is triggered when the incoming URL matches one of the values configured in the Logout URL List or the Agent Logout URL Regular Expression properties.

These pages must exist in your web server and should be the logout pages for your application.

If the incoming URL matches a logout URL, the agent creates a URL and redirects the web client to it. The URL contains:

- A logout page in your application or in AM. This page is configured in the OpenAM Logout URL property.

  If the Invalidate Logout Session property is enabled, the *agent* invalidates the session in AM. Configure this if the Logout URL List property is set to a page in your application, and your application *does not handle* the session invalidation process.

  If it is disabled, the logout page is responsible for invalidating the user session. Configure this if the Logout URL List property page is a SAML v2.0 logout page, the AM logout page, or a page in your application that can handle the session invalidation process.

- A `goto` parameter. Its value is the URL configured in the Logout Redirect URL property.

  Configure this if you want the user to end on a specific page of your application after logout. For example, the landing page, or a login page. This page must exist in your web server.

  If the Enable Logout URL Redirect property is disabled, the agent does not add the `goto` parameter, and the web client will remain in the logout page.

TIP

You can also configure the agent to reset specific cookies during logout by configuring the Logout Cookies List for Reset property.

Examples:

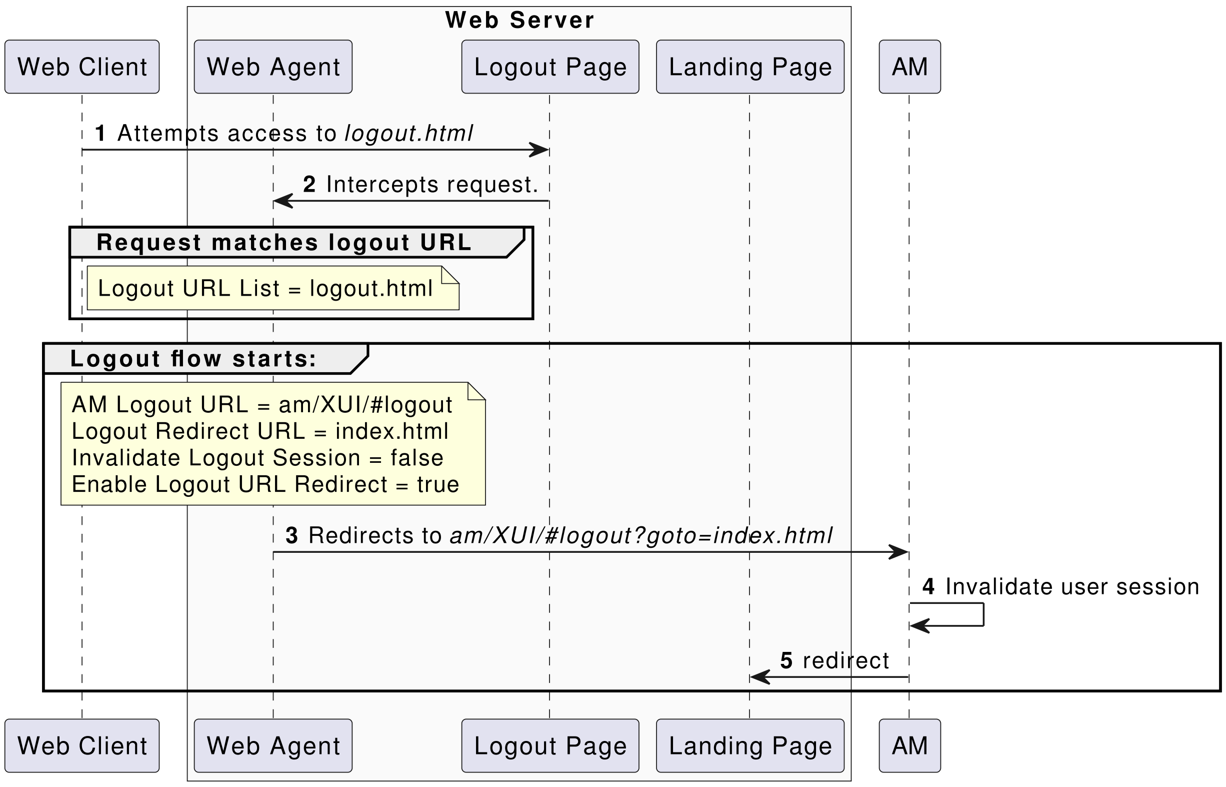

Web Server
Web Client
Web Agent
Logout Page
Landing Page
AM
1 Attempts access to logout.html
2 Intercepts request.
Request matches logout URL
Logout URL List = logout.html
Logout flow starts:
AM Logout URL = am/XUI/#logout
Logout Redirect URL = index.html
Invalidate Logout Session = false
Enable Logout URL Redirect = true
3 Redirects to am/XUI/#logout?goto=index.html
4 Invalidate user session
5 redirect

*Figure 1. Logout Flow with AM as the Logout Page*

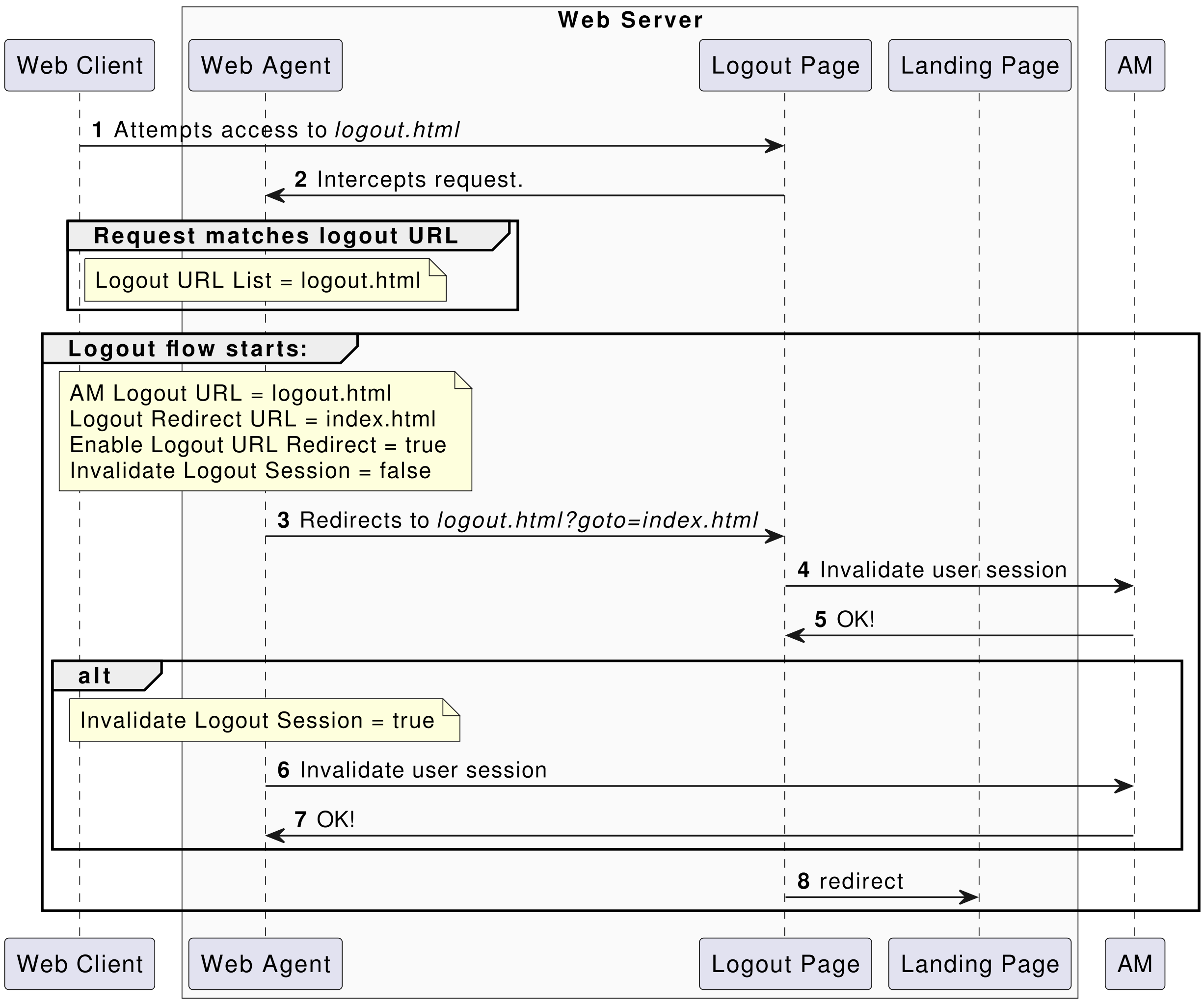

Web Server
Web Client
Web Agent
Logout Page
Landing Page
AM
1 Attempts access to logout.html
2 Intercepts request.
Request matches logout URL
Logout URL List = logout.html
Logout flow starts:
AM Logout URL = logout.html
Logout Redirect URL = index.html
Enable Logout URL Redirect = true
Invalidate Logout Session = false
3 Redirects to logout.html?goto=index.html
4 Invalidate user session
5 OK!
alt
Invalidate Logout Session = true
6 Invalidate user session
7 OK!
8 redirect

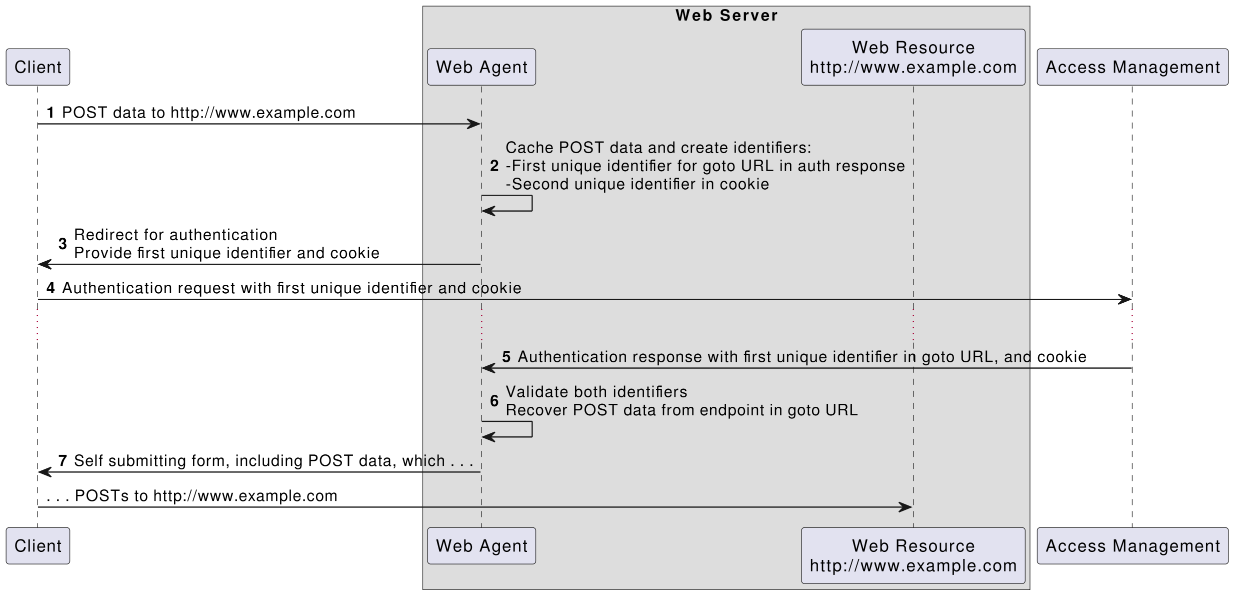
*Figure 2. Logout Flow with the Application Serving the Logout Page*

## POST Data Preservation

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

When POST data preservation is enabled, and an unauthenticated client posts HTML form data to a protected resource, the agent stores the data in its cache, and redirects the client to the login screen. After successful authentication, the agent recovers the data stored in the cache, and automatically submits it to the protected resource.

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



Web Server
Client
Web Agent
Web Resource http://www.example.com
Access Management
1 POST data to http://www.example.com
2 Cache POST data and create identifiers: -First unique identifier for goto URL in auth response -Second unique identifier in cookie
3 Redirect for authentication Provide first unique identifier and cookie
4 Authentication request with first unique identifier and cookie
5 Authentication response with first unique identifier in goto URL, and cookie
6 Validate both identifiers Recover POST data from endpoint in goto URL
7 Self submitting form, including POST data, which . . .
. . . POSTs to http://www.example.com

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

1. An unauthenticated client requests a POST to a protected resource.
2. The agent caches the POST data, and generates the following unique identifiers:
   - 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.

For information about configuration properties, see POST Data Preservation.

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

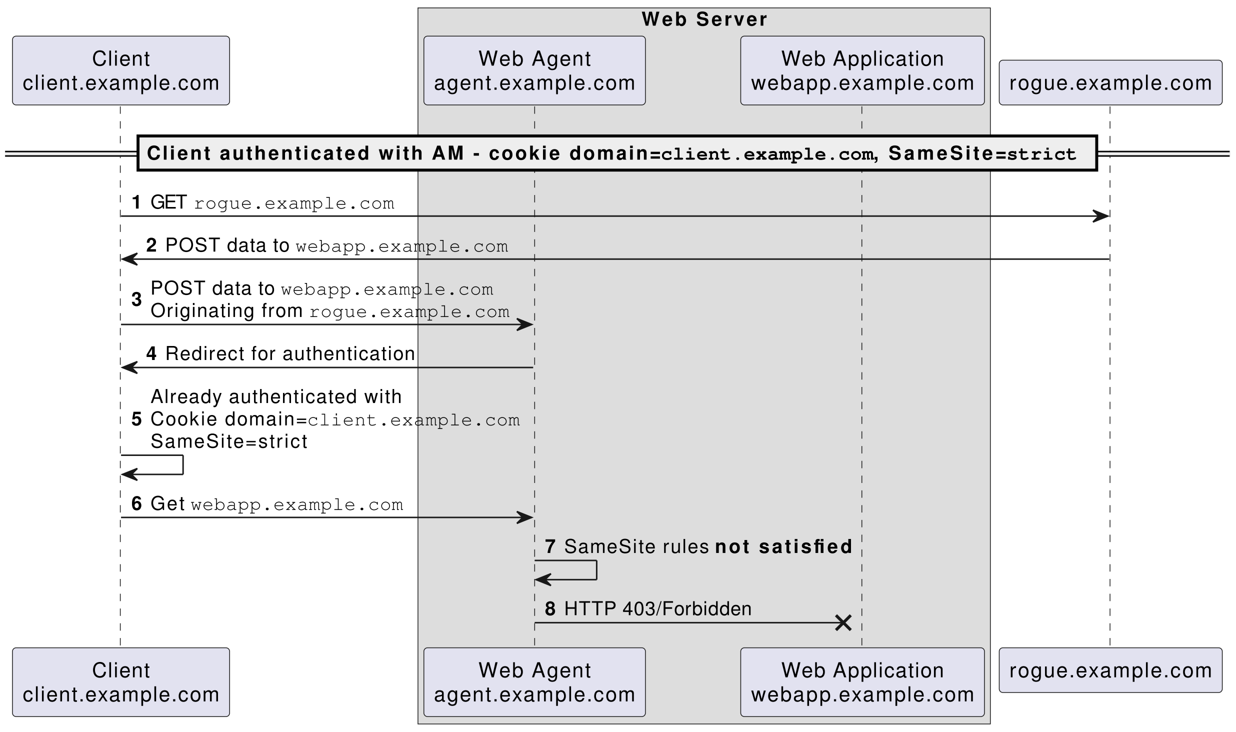
WARNING

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 Web Agent cannot provide generic protection against CSRF. ForgeRock recommends following the latest guidance from the OWASP CSRF Prevention Cheat Sheet.

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

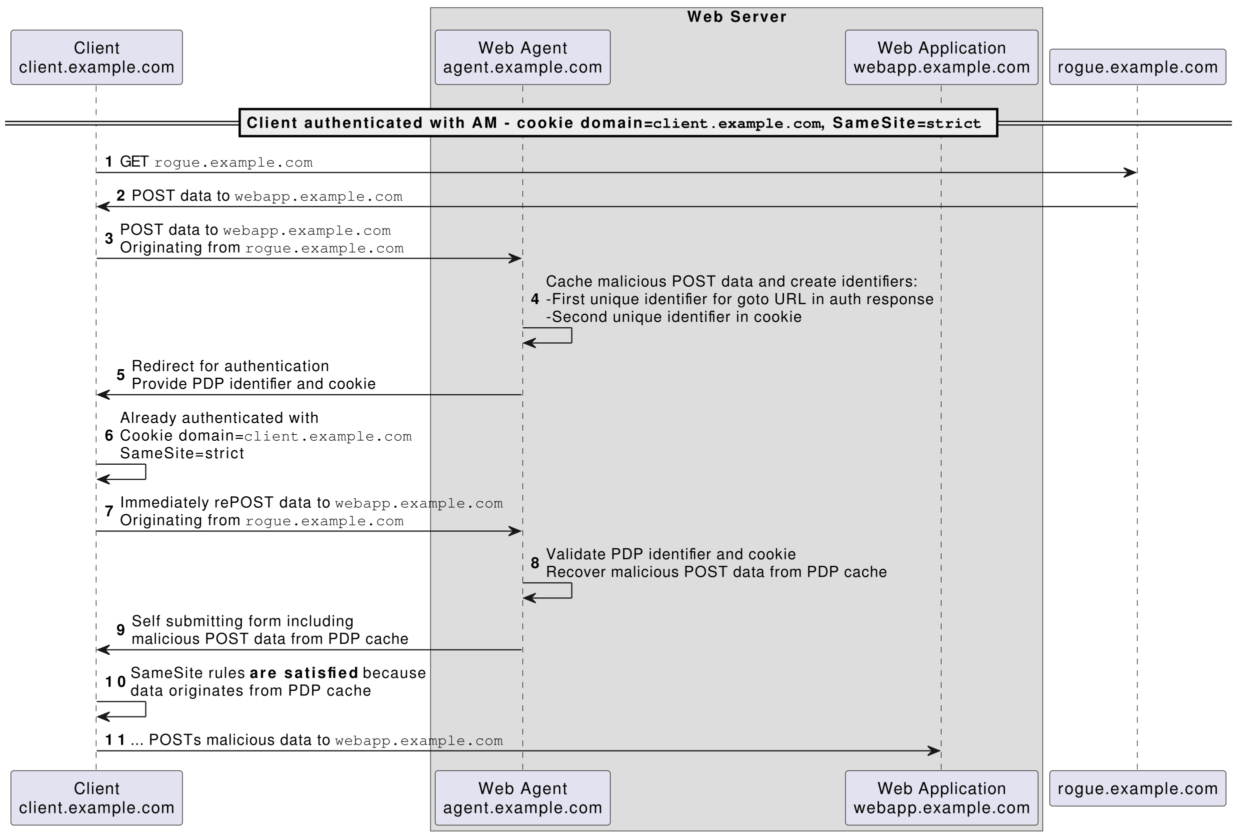
### *CSRF Attack When POST Data Preservation Is Disabled*

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


Web Server
Client client.example.com
Web Agent agent.example.com
Web Application webapp.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 Redirect for authentication
5 Already authenticated with Cookie domain=client.example.com SameSite=strict
6 Get webapp.example.com
7 SameSite rules not satisfied
8 HTTP 403/Forbidden
Client client.example.com
Web Agent agent.example.com
Web Application webapp.example.com
rogue.example.com

*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:


Web Server
Client client.example.com
Web Agent agent.example.com
Web Application webapp.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
Client client.example.com
Web Agent agent.example.com
Web Application webapp.example.com
rogue.example.com

## Caches

Web Agent supports the following caches to speed up agent operations:

### *Configuration Cache*

The configuration cache stores web agent configuration properties.

When a Web Agent starts up, it either makes a call to AM to retrieve a copy of the agent profile (centralized configuration mode) or reads the agent profile from the local configuration file (local configuration mode). Then, the agent stores the configuration in its cache. The cached information is valid until one of the following events occur:

- AM notifies the agent of changes to hot-swappable agent configuration properties. This only applies to deployments that use centralized configuration mode.
- The information in the cache reaches the expiration time specified by Configuration Reload Interval.

  When a configuration property in the cache is invalid, the agent clears the cached property value and rereads it from the agent profile.

### *Session and Policy Decision Cache*

Stored in the shared memory pool defined by the `AM_MAX_SESSION_CACHE_SIZE` environment variable, the session and policy decision cache stores session information, and the results of previous policy decisions.

The default size of the cache is 16 MB, but you may need to increase its size if you plan to hold many active sessions in the cache at any given time. For more information about the environment variable, see Environment Variables.

After authentication, AM presents the client with an ID token containing session information. The web agent stores part of that session information in the cache. When a client attempts to access a protected resource, the agent checks whether there is a policy decision cached for the resource:

- If there is a cached policy decision, the agent reuses it without contacting AM.
- If there is no cached policy decision, the validity of the client’s session determines the agent’s behavior:
  - If the client’s session is valid, the web agent requests a policy decision from AM, caches it, and then enforces it.
  - If the client’s 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, and the session is cached, the agent requests a policy decision from AM, caches it, and then enforces it.

Session and policy decisions are valid in the cache until one of the following events occur:

*Session and Policy Decision Validity in Cache*

| Event | What is invalidated? |
|---|---|
| Session contained in the ID token 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 |
| Session reaches the expiration time specified by SSO Cache Polling Period. | Session |
| Policy decision reaches the expiration time specified by Policy Cache Polling Period. | Policy decision |
| Administrator makes a change to policy configuration (and policy notifications are enabled) | All sessions and all policy decisions |

IMPORTANT

A Web Agent that loses connectivity with 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(*).

## *Policy Cache*

The policy cache builds upon the session and policy decision cache. It downloads and stores details about policies from AM, and uses the downloaded policies to make authorization decisions, without contacting AM each time.

Web Agent uses the policy cache without contacting AM in the following situations:

- A requested resource matches the resource pattern of a policy that has been cached due to a previous evaluation.
- A requested resource **does not** match any cached policy patterns. In this case, the agent denies access immediately.
- A requested resource matches the resource pattern of a simple policy that applies to the `All Authenticated Users` virtual group.

  If the resource matches the policy used for a previous policy decision, the agent does not request policy evaluation from AM. Therefore, policy conditions based on scripts, LDAP filter conditions, or session properties, which rely on attributes that can vary during a session, may not be enforced.

To reduce this risk, you should:

- Enable session property change notifications, as described in Notifications.
- Reduce the amount of time that sessions can remain in the agent session cache.

The following caveats apply when using the policy cache:

- If you have a large number of policies, for example more than one million in an UMA deployment, the time to download the policies and the memory consumption of the agent may affect performance.
- The agent downloads the policy rules, and uses them to evaluate policies locally. If a policy is customized in AM in a way that changes the way it is evaluated (for example, a wildcard or delimiter is changed), the policy decision made by the agent might not match the policy defined in AM.
- Even though delimiters and wildcards are configurable in AM (Configure > Global Services > Policy Configuration > Global Attributes > Resource Comparator), the policy cache only supports the default configuration.

Do not enable the agent's policy cache if your policies use custom delimiters and/or wildcards.

Enable the policy cache by creating an environment variable named `AM_POLICY_CACHE_MODE`.

Change the location of the policy cache by creating an environment variable named `AM_POLICY_CACHE_DIR`.

For more information about properties related to the policy cache, see Environment Variables.

### *POST Data Preservation Cache*

Stored in files saved in the agent installation directory, the POST data preservation cache stores short-lived POST data.

When Enable POST Data Preservation is enabled, the agent caches HTML form data submitted as an HTTP POST by unauthenticated clients. By default, this data is stored in the directory specified by POST Data Storage Directory.

POST data information is cached for the amount of time specified by POST Data Entries Cache Period.

## Connection Pooling

By default, the Web Agent uses connection pooling to improve performance when AM is available over low bandwidth connections, or to throttle the maximum number of connections made by the Web Agent.

When AM is available over high bandwidth connections, connection pooling can reduce performance.

Enable and disable connection pooling with the bootstrap property Enable Connection Pooling.

## Configuration for Load Balancers and Reverse Proxies

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

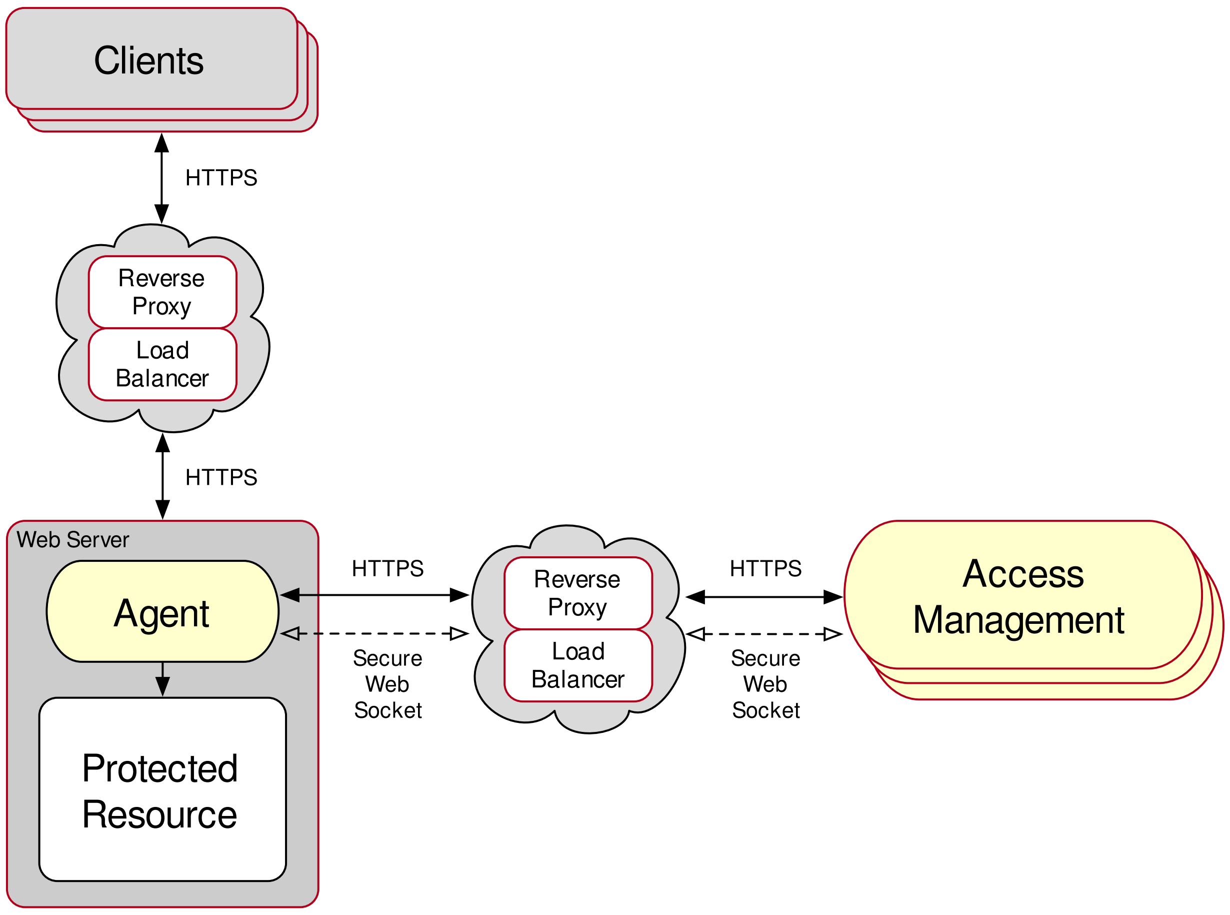


*Figure 3. Environments with Load Balancers and Reverse Proxies*

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 Connection to AM Through a Load Balancer/Reverse Proxy

When a reverse proxy is situated between the agent and AM, it can be used to protect the AM APIs.

When a load balancer is situated between the agent and AM, it can be used to regulate the load between different instances of AM.

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

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

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

To prevent problems during installation or redirection, do the following:

- 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 agents are configured with multiple agent profiles, configure sticky load balancing. The agent profile name is contained in the OpenID Connect JWT, used by the agent and AM for communication. Without session stickiness, it is not possible to make sure that the appropriate JWT ends in the appropriate agent instance.

  To have multiple agent profiles without sticky load balancing, disable validation of the `aud` claim in the session ID token. Either enable Disable Audience Claim Validation, or configure Agent Profile ID Allow List.

  For security reasons, agents should validate all claims in session ID tokens. Therefore, use this approach sparingly and mostly for migrations.
- 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 Configuring Apache HTTP Server as a Reverse Proxy Example.

### *Configuring 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.

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

1. Log in to the AM console as an administrative user, such as `amAdmin`.
2. Select Realms > Realm Name > Services.
3. Select Add a Service > Base URL Source, and create a default service, leaving the fields empty.
4. Configure the service with the following properties, leaving the other fields empty:
   - **Base URL Source**: `X-Forwarded-* headers`

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

- **Context path**: AM’s deployment URI. For example, `/openam`.

For more information, see Base URL Source in AM’s *Reference*.

5. Save your changes.

## Agent Connection to Clients Through a Load Balancer/Reverse Proxy

When a reverse proxy is situated between the agent and client, it can be used to anonymize the client traffic that enters the network.

When a load balancer is situated between the agent and client, it can be used to regulate the load between the agents and the web application servers.

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

### *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.

### *POST Data Preservation and Sticky Load Balancing*

For POST data preservation, use sticky load balancing to ensure that after login the client hits the same agent as before and can therefore get 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 Configure POST Data Preservation for Load Balancers or Reverse Proxies.

### *Mapping Agent Host Name to a Load Balancer or Reverse Proxy*

In the following diagram, the agent and load balancer/reverse proxy use the same protocol and port, but different FQDNs:

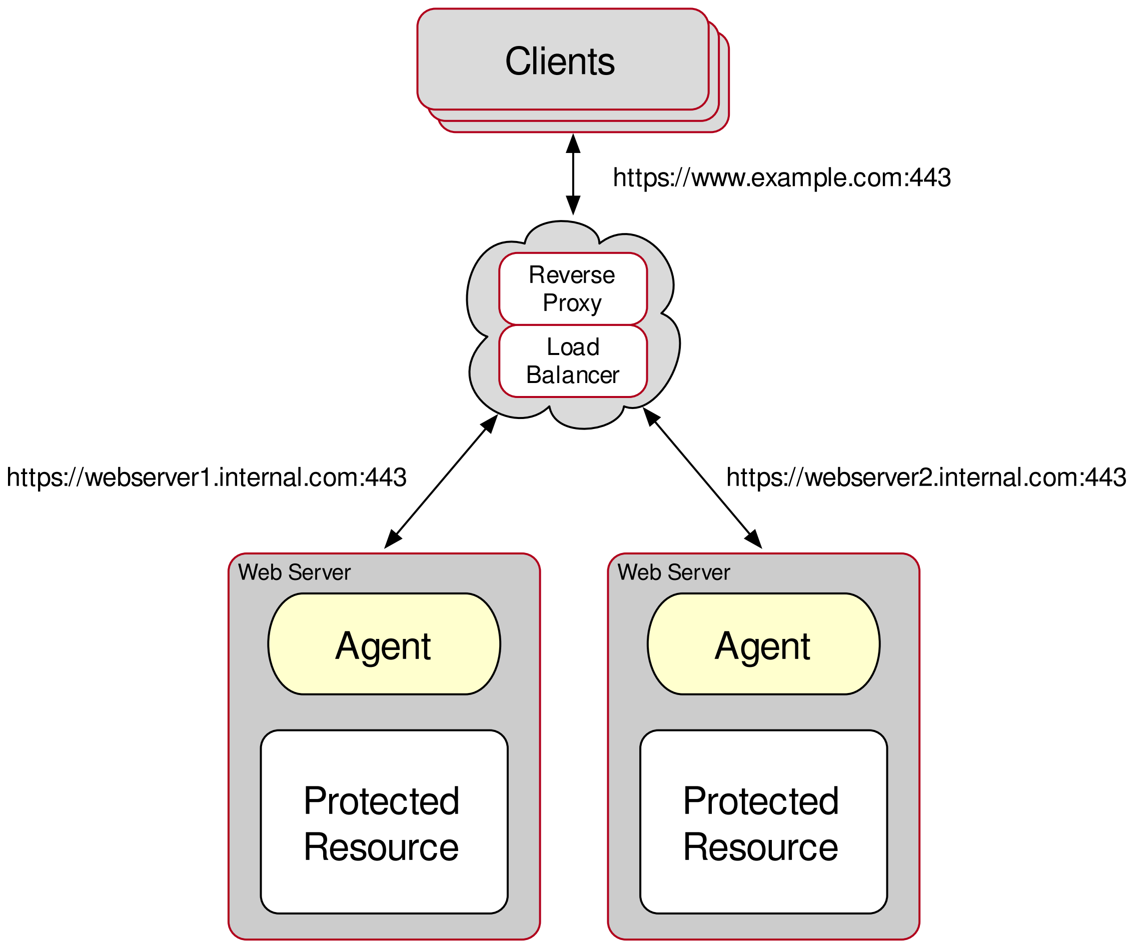


*Figure 4. Same Protocol and Port, Different FQDN*

Map the host name of the agent to that of the load balancer or reverse proxy.

1. Log in to the AM console as an administrative user with rights to modify the web agent profile.
2. Go to Realms > Realm Name > Applications > Agents > Web > Agent Name.
3. Set the following options in the Global tab:
   - **FQDN Check**: Enable

     The equivalent property setting is Enable FQDN Check = `true`.
   - **FQDN Default**: Set to the FQDN of the load balancer or proxy, for example `lb.example.com`. Do not set it to the protected server FQDN where the agent is installed.

     The equivalent property setting is FQDN Default = `lb.example.com`.
   - **Agent Root URL for CDSSO**: Set to the FQDN of the load balancer or proxy, for example `https://lb.example.com:80/`.

The equivalent property setting is Agent Root URL for CDSSO = `lb.example.com`.

- **FQDN Virtual Host Map**: Map the load balancer or proxy FQDN to the FQDN where the agent is installed. For example,
  - **Key**: `agent.example.com` (protected server)
  - **Value**: `lb.example.com` (load balancer or proxy)

    The equivalent property setting is `com.sun.identity.agents.config.fqdn.mapping[agent.example.com]=lb.example.com`.

4. Save your work.

### *Overriding Protocol, Host, and Port*

The load balancer or reverse proxy forwards requests and responses between clients and protected web servers. In the following diagram, the protocol, port, and FQDN configured on the load balancer and reverse proxy are different than those on the protected web server:

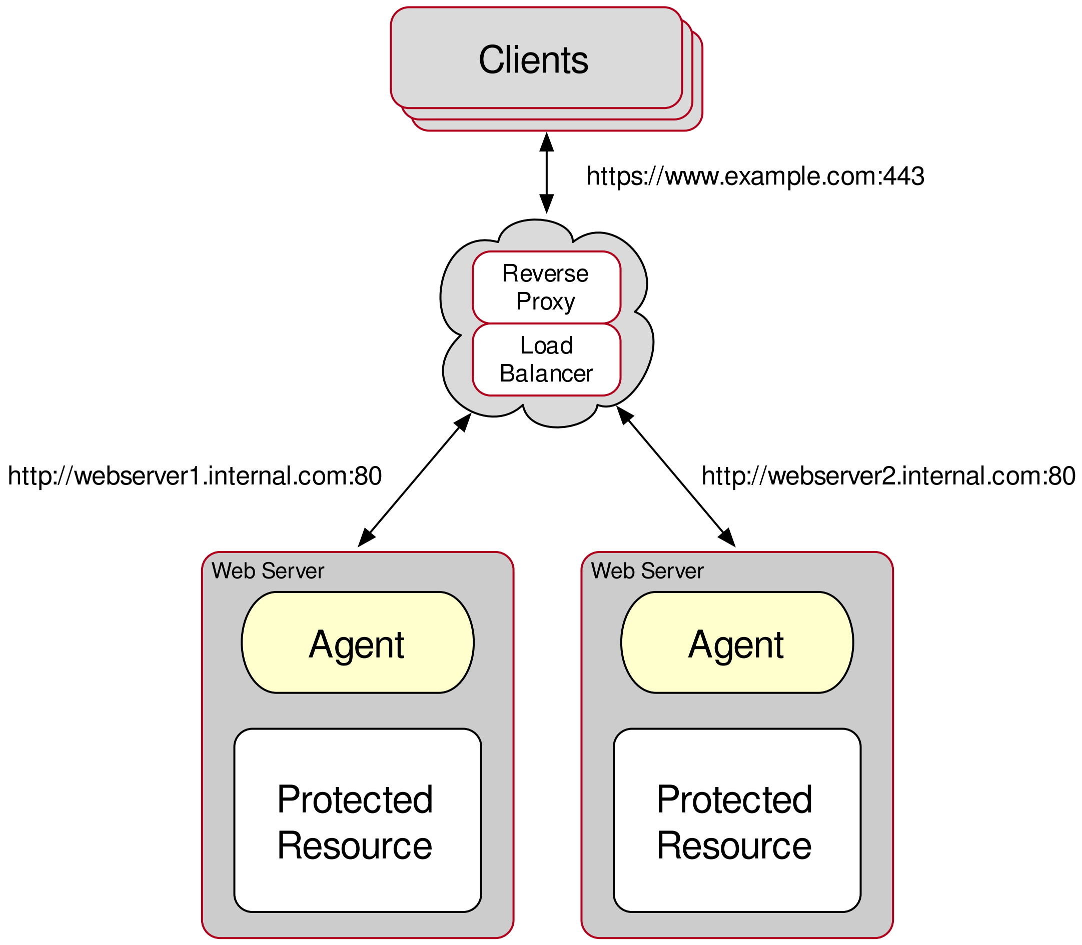


*Figure 5. Different Protocol, Port, and FQDN*

IMPORTANT

IMPORTANT

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

NOTE

When the following headers are defined on the proxy or load-balancer, they override the value of Agent Deployment URI Prefix:

- `X-Forwarded-Proto`
- `X-Forwarded-Host`
- `X-Forwarded-Port`

If you are using these headers, do not configure the agent to override its hostname, port, or protocol.

Use Agent Deployment URI Prefix to override the agent protocol, host, and port with that of the load balancer or reverse proxy.

1. Log in to the AM console as an administrative user with rights to modify the agent profile.
2. Go to Realms > Realm Name > Applications > Agents > Web > Agent Name.
3. Set the following options in the Global tab:
   - **Agent Deployment URI Prefix**: Set to the URI of the load balancer or proxy. For example, `https://external.example.com:443`.

     This value is used to override the protocol, host, and port of the protected server.

     The equivalent property setting is Agent Deployment URI Prefix = `https://external.example.com:443`.
   - **Agent Root URL for CDSSO**: Set to the FQDN of the load balancer or proxy, for example `https://lb.example.com:80/`.

     The equivalent property setting is Agent Root URL for CDSSO = `lb.example.com`.
4. Enable the following options in the Advanced tab:
   - **Override Request URL Protocol**

     The equivalent property setting is Enable Override Request URL Protocol = `true`.

- **Override Request URL Host**

  The equivalent property setting is Enable Override Request URL Host = `true`.

- **Override Request URL Port**

  The equivalent property setting is Enable Override Request URL Port = `true`.

5. Save your work.

### *Configuring 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 for a centralized web agent profile configured in the AM console. The steps also mention the properties for web agent profiles that rely on local, file-based configurations:

1. Log in to the AM console with a user that has permissions to modify the web agent profile.
2. Go to Realms > Realm Name > Applications > Agents > Web > Agent Name.
3. Set the following options in the Advanced tab:
   - **Client IP Address Header**: Configure the name of the header containing the IP address of the client. For example, `X-Forwarded-For`.

     The equivalent property setting is Client IP Address Header = `X-Forwarded-For`.

     Configure this property if any of the following points are true:

     - AM policies are IP address-based.
     - The agent is configured for not-enforced IP rules.
     - The agent is configured take any decision based on the client's IP address.
   - **Client Hostname Header**: Configure the name of the header containing the FQDN of the client. For example, `X-Forwarded-Host`.

     The equivalent property setting is Client Hostname Header = `X-Forwarded-Host`.

     Configure this property if any of the following points are true:

     - AM policies are URL address-based.

        - The agent is configured for not-enforced URL rules.
        - The agent is configured take decisions based on the client's URL address.
4. Save your changes.

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

Use one of the following procedures to configure post data preservation for load balancers or reverse proxies.

#### *Map One Agent Profile to One Agent Instance When POST Data Preservation is Enabled*

In this procedure, a separate agent profile must created in AM for each agent instance. For scalable deployments, where resources are dynamically created and destroyed, use Map One Agent Profile to Multiple Agent Instances When POST Data Preservation is Enabled instead.

1. Configure your load balancer or reverse proxy to ensure session stickiness when the cookie or URL query parameter are present.
2. Log in to the AM console as a user that has permissions to modify the agent profile.
3. Go to **REALMS** > Realm Name > **Applications** > **Agents** > **Web** > Agent Name.
4. Set the following options in the Advanced tab:
    - POST Data Sticky Load Balancing Mode:
        - **COOKIE**: The agent creates a cookie for POST data preservation session stickiness. The content 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 `com.sun.identity.agents.config.postdata.preserve.stickysession.mode=COOKIE` or `com.sun.identity.agents.config.postdata.preserve.stickysession.mode=URL`.
    - POST Data Sticky Load Balancing Value: Configure a key-pair value separated by the `=` character.

      The agent creates the value when it evaluates POST Data Sticky Load Balancing Mode. For example, specifying `lb=myserver` either sets a cookie called `lb` with `myserver` as a value, or appends `lb=myserver` to the URL query string.

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

5. Save your changes.

*Map One Agent Profile to Multiple Agent Instances When POST Data Preservation is Enabled*

Use this procedure for scalable deployments, where resources can be dynamically created or destroyed. For example, use it in deployments with load balancing, or environments running Kubernetes.

1. Configure your load balancer or reverse proxy to ensure session stickiness when the cookie or URL query parameter are present.
2. For each agent instance, configure post data preservation in the agent configuration file `agent.conf`. This configuration overrides the configuration in AM.

   In the following example, the settings in `agent.conf` configure two agents behind a load balancer to use the same agent profile, and provide uniqueness to the load balancer:

   - Agent 1:

     ```
     com.sun.identity.agents.config.postdata.preserve.sticky
     session.mode = COOKIE
     com.sun.identity.agents.config.postdata.preserve.sticky
     session.value = EXAMPLE=Agent1
     ```

   - Agent 2:

     ```
     com.sun.identity.agents.config.postdata.preserve.sticky
     session.mode = COOKIE
     com.sun.identity.agents.config.postdata.preserve.sticky
     session.value = EXAMPLE=Agent2
     ```

   For information about the values to use, see the following properties:

   - POST Data Sticky Load Balancing Mode
   - POST Data Sticky Load Balancing Value
3. Restart the web server where the agent is installed.

## Troubleshooting

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

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

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

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

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

The following items are some common issues and solutions:

TIP

The **`agentadmin`** command offers a validation mode for the agent that can help you troubleshoot issues in your environment; for example, after an agent upgrade or a network change. For more information, see the `--V[i]` option in agentadmin Command.

▼ Error installing Agents on Windows

***Question***

I am trying to install a web agent on Windows, which will connect to an AM server running over HTTPS, but the installer reports the following error:

```
init_ssl(): ssleay32.dll is not available (error: 87)
init_ssl(): libeay32.dll is not available (error: 87)
```

*Answer*

If OpenSSL is correctly installed, on Windows 7 or Windows Server 2008 R2 systems, apply the update provided in Microsoft knowledge base article KB2533623. See Microsoft Security Advisory: Insecure library loading could allow remote code execution.

▼ Error installing Agents With SELinux

*Question*

I am trying to install Web Agent on a server with SELinux enabled in `enforcing` mode and I am getting error messages after installation, or the web server does not start up. What happened?

*Answer*

When installing Web Agent on Linux or Unix servers, you must ensure that the user that runs the web server process has read and write permissions for the agent installation directory and files.

If SELinux is enabled in `enforcing` mode, you must also ensure that SELinux is configured to allow the web server process to perform read and write operations to the agent installation directory and files. By default, SELinux only allows the web server process to read files in well-known authorized locations, such as the `/var/www/html` directory.

For environments where security can be more relaxed, consider setting SELinux or the `httpd_t` context in `permissive` mode for troubleshooting purposes.

For information about configuring SELinux, see the Linux documentation.

▼ Logs Not Written

*Question*

Why are logs not being written to `/log/system_0.log` and `/log/monitor_0.pipe` files? I am seeing this error:

```
unable to open event channel
```

*Answer*

It is likely that the agent does not have permission to be able to write to the `/log/system_0.log` and `/log/monitor_0.pipe` log files.

This can occur if you used the **`agentadmin --V[i]`** validator command using a user account that is different to the account used to run your web server.

Run the validator command as the same user that runs the web server, for example, by using the **`sudo`** command.

To fix the issue, change the ownership of these files to match the user or group that is running your web server.

▼ Port 80 Used as Default

***Question***

My Apache HTTP server is not using port 80. When I install Web Agent it defaults to port 80. How do I fix this?

***Answer***

You probably set `ServerName` in the Apache HTTP Server configuration to the host name, but did not specify the port number.

Instead, set both the host name and port number for `ServerName` in the configuration. For example, if you have Apache HTTP Server configured to listen on port 8080, then set `ServerName` appropriately as in the following excerpt:

```
<VirtualHost *:8080>
ServerName www.localhost.example:8080
```

▼ Cannot rotate logs

***Question***

My web server and Web Agent are installed as root, and the agent cannot rotate logs. I am seeing this error:

```
Could not rotate log file ... (error: 13)
```

What should I do?

***Answer***

If the web server is running with a non-root user, for example, the `daemon` user, you must ensure that user has the following permissions:

- Read Permission:
  - `/web_agents/agent_name/lib`
- Read and Write Permission:
  - `/web_agents/agent_name/instances/agent_nnn`
  - `/web_agents/agent_name/log`

Apply execute permissions on the folders listed above, recursively, for the user that runs the web server.

For IIS web agents, change the ownership of the files using the **`agentadmin --o`** command. For more information, see agentadmin Command.

> TIP
>
> You may also see similar issues if SELinux is enabled in `enforcing` mode, and it is not configured to allow access to agent directories.

▼ Protection Against Phishing Attacks

*Question*

How do I increase security against possible phishing attacks through open redirect?

*Answer*

You can specify a list of valid URL resources against which AM validates the `goto` and `gotoOnFail` URL using the Valid `goto` URL Resource service.

AM only redirects a user if the `goto` and `gotoOnFail` URL matches any of the resources specified in this setting. If no setting is present, it is assumed that the `goto` and `gotoOnFail` URL is valid.

To set the Valid `goto` URL Resources, use the AM console, and go to Realms > Realm Name > Services. Click Add, select Validation Service, and then add one or more valid `goto` URLs.

You can use the "*" wildcard to define resources, where "*" matches all characters except "?". For example, you can use the wildcards, such as `https://website.example.com/*` or `https://website.example.com/*?*`. For more specific patterns, use resource names with wildcards as described in Configuring Success and Failure Redirection URLs.

▼ Apache Agent Start Up

*Question*

I have installed the Unix Apache Web Agent, and neither Apache nor the agent start up or log any message. If I remove the agent, the Apache server starts again. What can be the problem?

*Answer*

To troubleshoot Web Agent or a web server that does not start, set the agent logging level to the maximum by performing the following steps:

1. Set the environment variable `AM_SYSTEM_LOG_LEVEL` to `All` in your command line session. For example:

```
$ export AM_SYSTEM_LOG_LEVEL=ALL
```

2. Restart the Apache server.
3. Check the logs generated in the `web_agent/apache24_agent/log/system_n.log`.

   Web Agent reserves memory for the policy and session cache based on the `AM_MAX_SESSION_CACHE_SIZE` environment variable. If the server where the agent is installed does not have enough shared memory available, the web agent may log messages like the following:

   ```
   017-11-10 12:06:00.492 +0000   DEBUG [1:7521]
   [source/shared.c:1451]am_shm_create2() about to create
   block-clusters_0, size 1074008064
   2017-11-10 12:06:00.492 +0000   ERROR
   [1:7521]am_shm_create2(): ftruncate failed, error: 28
   ```

   The error message means the web agent tries to reserve 1074008064 bytes of memory, but there is not enough shared memory available. Several reasons may explain why the shared memory is running low, such as:

   - A new application or additional workload may be stretching the server resources to the limit.

     In this case, ensure that the server has enough shared memory available to satisfy the need of all the applications.

   - A web agent may not have been able to release its shared memory after stopping. Therefore, even if the shared memory is technically not in use, it is still reserved and cannot be reassigned unless freed.

     Different operating systems manage the shared memory in different ways. Refer to your operating system documentation for information about checking shared memory usage.

     You can reduce the amount of memory the web agent reserves for the session and policy cache by setting the `AM_MAX_SESSION_CACHE_SIZE` environment variable to a value between 1048576 (1 MB) and 1074008064 bytes (1 GB). For more information, see Environment Variables.

     Troubleshooting a component that does not start and does not generate logs may be difficult to diagnose. Contact the ForgeRock Support team for more help and information.

▼ Infinite Redirection Loops

***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:

```
... +0000 ERROR [c5319caa-beeb-5a44-a098-d5575e768348]state
identifier not present in authentication state
2018-03-15 16:23:10.538 +0000 WARNING [c5319caa-beeb-5a44-
a098-d5575e768348]unable to verify pre-authentication cookie
... +0000 WARNING [c5319caa-beeb-5a44-a098-
d5575e768348]convert_request_after_authn_post(): unable to
retrieve pre-authentication request data
... +0000 DEBUG [c5319caa-beeb-5a44-a098-d5575e768348] exit
status: forbidden (3), HTTP status: 403, subrequest 0
```

What is happening?

***Answer***

The redirection loop happens because the client-based (stateless) session cookie is surpassing the maximum supported browser header size. Since the cookie is incomplete, AM cannot validate it.

To ensure the session cookie does not surpass the browser supported size, configure either signing and compression or encryption and compression.

For more information, see AM's Security Guide.

▼ Errors After Upgrade

***Question***

I have upgraded my agent and, in the logs, I can see errors similar to the following:

```
redirect_uri_mismatch. The redirection URI provided does not
match a pre-registered value.
com.iplanet.sso.SSOException: Invalid Agent Root URL
com.iplanet.sso.SSOException: Goto URL not valid for the agent
Provider ID
```

What should I do?

***Answer***

Web Agent accepts 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, Web Agent prevents you from accessing the agent on URLs not defined in the Agent Root URL for CDSSO property. Add entries to this property when:

- Accessing the agent through different protocols. For example, `http://agent.example.com/` and `https://agent.example.com/`.
- Accessing the agent through different virtual host names. For example, `http://agent.example.com/` and `http://internal.example.com/`.

- Accessing the agent through different ports. For example, `http://agent.example.com/` and `http://agent.example.com:8080/`.

## ▼ Configuration Not Updated After Upgrade

*Question*

I have upgraded my Unix Apache or IBM HTTP Server Web Agent, and even though notifications are enabled, the agent does not update its configuration. What is happening?

*Answer*

Set the web agent logging level to the maximum by performing the following steps:

1. Set the environment variable `AM_SYSTEM_LOG_LEVEL` to `ALL` in your command line session. For example:

   ```
   $ export AM_SYSTEM_LOG_LEVEL=ALL
   ```

2. Restart the Apache or IBM HTTP server.

3. Check the logs generated in the `web_agent/agent_type/log/system_n.log` file.

   Sometimes stopping or upgrading an agent does not clean the pipe file the agent uses to communicate with AM. If the newly started agent cannot create the pipe to communicate with AM because it already exists, the agent would log messages like the following:

   ```
   ... UTC   DEBUG [1:10551398][source/monitor.c:503]monitor
   startup
   ... UTC   ERROR [102:10551398]monitor unable to get
   semaphore
   ... UTC   DEBUG [304:10551398]
   [source/config.c:295]config_initialise():  agent
   configuration read from cache, agent: / wpa-aix7-Httpd7-
   32bit
   ```

If you see similar error messages, perform the following steps to delete the pipe file:

1. Stop the Apache or IBM HTTP server.
2. Change directories to the `/tmp` directory.
3. Delete the `monitor.pipe` file.
4. Restart the Apache or IBM HTTP server.

## ▼ Custom Pages Not Displayed

*Question*

After upgrade, the default Apache welcome page appears instead of my custom error pages. What should I do?

***Answer***

Check your Apache `ErrorDocument` configuration. If the custom error pages are not in the document root of the Apache server, you should enclose the `ErrorDocument` directives in `Directory` elements. For example:

```
<Directory "/web/docs">
   ErrorDocument 403 myCustom403Page.html
</Directory>
```

For more information about `ErrorDocument`, see the Apache documentation.

▼ Failure After Installation

***Question***

After starting a web agent installation, I see a failure in the logs:

```
[../resources/troubleshooting/troubleshooting.bash:#web-agent-
install]
```

***Answer***

Web Agent installation, can fail if AM's validation of the agent configuration exceeds the default timeout of 4 seconds.

You can set the `AM_NET_TIMEOUT` environment variable to change the default timeout, and then rerun the installation.

▼ Agent Not Protecting a Website

***Question***

My Web Agent is not protecting my website. In the logs, I can see errors similar to the following:

```
... -0500  ERROR [86169084-5648-6f4d-a706-
30f5343d9220]config_fetch():  failed to load configuration for
agent: myagent myagent, error -24
... -0500  ERROR [86169084-5648-6f4d-a706-
30f5343d9220]amagent_auth_handler(): failed to get agent
configuration instance, error: invalid agent session*
```

What is happening?

***Answer***

The Web Agent is unable to log in to AM. Possible causes are:

- Network connection between the agent and AM is unavailable.
- The AM Connection URL property, which specifies the AM URL may be misconfigured.

▼ Agent Not Protecting a Website

***Question***

My Web Agent is not protecting my website. In the `debug.log` file I can see messages similar to the following:

```
... GMT DEBUG [162ba6eb-cf88-3d7f-f92c-ee8b21971b4c]:
(source/oidc.c:265) agent_realm does not have the expected
value: JWT
    {
    "sub":"demo",
    "auditTrackingId":"267d1f56-0b97-4830-ae91-6be4b8b7099f-
5840",

"iss":"https://openam.example.com:8443/openam/oauth2/Customers",
    "tokenName":"id_token",
    "nonce":"D3AE96656D6D634489AF325D90C435A2",
    "aud":"webagent",
    "s_hash":"rxwxIoqDFiwt4MxSwiBa-w",
    "azp":"webagent",
    "auth_time":1561600459,
    "forgerock":{
    "ssotoken":"wi8tHql...MQAA*",
    "suid":"267d1f56-0b97-4830-ae91-6be4b8b7099f-5647"
    },
    "realm":"/Customers",
    "exp":1561607661,
    "tokenType":"JWTToken",
    "iat":1561600461,
    "agent_realm":"/Customers"
    }
... GMT WARNING [162ba6eb-cf88-3d7f-f92c-ee8b21971b4c]:
redirect_after_authn(): unable to validate JWT
```

What is happening?

***Answer***

If you configured the agent profile in a realm other than AM's top-level realm ( / ), you must configure the agent `com.sun.identity.agents.config.organization.name` bootstrap property with the realm where the agent profile is located. For example, `/Customers`.

Realm names are case-sensitive. Failure to set the realm name exactly as configured in AM causes the agent to fail to recognize the realm.

▼ Cannot Access Protected Resources

I am getting HTTP 403 Forbidden messages when accessing protected resources, and I can see errors similar to the following in the `debug.log` file:

```
... GMT WARNING [69d4632c-82af-b853-0f340vb7b754]: too many
pending authentications
... GMT ERROR  [69d4632c-82af-76da-b853-0f340vb7b754]:
save_pre_authn_state(): unable to save state for request
```

What is happening?

***Answer***

Agents store the progress of authentication with AM in the pre-authentication cookie, `agent-authn-tx`. This cookie has a maximum size of 4096 bytes, and can fill up if the agent receives many parallel unauthenticated requests to access protected resources.

For more information, see Enable Multivalue for Pre-Authn Cookie.

▼ Cannot Access Protected Resources

***Question***

I am getting HTTP 403 Forbidden messages when accessing the Web Agent.

***Answer***

Make sure that the Web Agent is executable:

1. In the terminal where the Web Agent is running, go to `/opt/web_agents`.
2. Review and, if necessary, change the permissions for the directory:

# Reference

## Environment Variables

This section describes Web Agent properties that are configured by environment variables. After setting an environment variable, restart the container where Web Agent is running.

Configure environment variables to affect the user that is running the web server, virtual host, or location that the agent protects.

TIP

For information about allowing environment variables to be used in NGINX, see the env directive in the *NGINX Core functionality documentation*.

***AM_IPC_BASE***

(Unix only) The base number for IPC identifiers used by the agent. The shared memory semaphore ID range used by the agent starts at the specified value. Set this variable only if you detect that the agent semaphores are clashing with those of other processes in your environment.

The default is an arbitrary value.

***AM_MAX_AGENTS***

The maximum number of agent instances in the installation. The higher the number, the more shared memory the agent reserves. The default value is `32`.

Once the number is met, any additional agent instances that start will log an error, and will not protect resources.

***AM_MAX_SESSION_CACHE_SIZE***

The maximum size of the shared memory for the session and policy cache, in bytes:

- Not set, or set to `0`: `16777216` (16 MB)
- Maximum value: `1073741824` (1 GB)
- Minimum value `1024` (1 MB)

For multiple concurrent sessions, consider using a higher value.

***AM_NET_TIMEOUT***

The number of seconds for which the agent installer can contact AM during agent configuration validation.

If the installer takes longer than this value to contact AM and validate the configuration, installation fails.

Default: 4 seconds

***AM_POLICY_CACHE_MODE***

Set to `on` to enable the policy cache.

You must also specify a directory in which to store the policies in the `AM_POLICY_CACHE_DIR` environment variable.

***AM_POLICY_CACHE_DIR***

The directory in which to store the policy cache. The agent must be able to write to this directory. For example, `/path/to/web_agents/<agent_type>/log`.

***AM_RESOURCE_PERMISSIONS***

(Unix only) The permissions that the agent sets for its runtime resources. Possible values are:

- `0600`
- `0660`
- `0666`

The `AM_RESOURCE_PERMISSIONS` environment variable requires the `umask` value to allow these permissions for the files.

Consider an example where the Apache agent is running with the `apache` user. The `umask` value is set to `0022` and the `AM_RESOURCE_PERMISSIONS` environment variable is set to `0666`. The agent runtime resources will have the following permissions:

*Resource Permissions Example in Linux*

| Resource | Permission | Owner |
|---|---|---|
| `/path/to/web_agents/agent_type/log/system_n.log` | 644 | apache |
| `/path/to/web_agents/agent_type/log/monitor_n.log` | 644 | apache |
| `/path/to/web_agents/agent_type/instances/agent_n/conf/agent.conf` | 640 | apache |
| `/path/to/web_agents/agent_type/instances/agent_n/logs/debug/debug.log` | 644 | apache |
| `/dev/shm/am_cache_0` | 644 | apache |
| `/dev/shm/am_log_data_0` | 644 | apache |

Any semaphores owned by the `apache` user have `644` permissions as well.

Consider another example where `umask` is set to `0002` and the` AM_RESOURCE_PERMISSIONS` environment variable is set to `0666`. The files would be created with `664` permissions, which would allow the files to be read and written by the members of the group, as well.

***`AM_SSL_OPTIONS`***

Overrides the default SSL/TLS protocols for the agent, set in the Security Protocol List bootstrap property.

Specifies a space-separated list of security protocols preceded by a dash ( `-` ) that will *not* be used when connecting to AM.

The following protocols are supported:

- `SSLv3`
- `TLSv1`
- `TLSv1.1`
- `TLSv1.2` (Enabled)
- `TLSv1.3` (Enabled)

For example, to configure `TLSv1.1`, set the environment variable to `AM_SSL_OPTIONS = -SSLv3 -TLSv1 -TLSv1.2`.

***AM_SYSTEM_LOG_LEVEL***

The log level of garbage collector statistics for all Web Agent instances in the container. The logs are written into the `system_n.log` file, where `n` indicates the agent group number.

Consider an environment with two Apache server installations:

- `Apache_1` has two agent instances configured, `agent_1` and `agent_2`, configured to share runtime resources (AmAgentId is set to 0). Both agent instances will write to the `system_0.log` file.
- `Apache_2` has one agent instance configured, `agent_3`, with AmAgentId set to 1. The instance will write to the `system_1.log` file.

By default, the `system_n.log` file is stored in the `/path/to/web_agents/agent_type/log` directory. To modify its path or its size, configure the `AM_SYSTEM_LOG_PATH` and `AM_SYSTEM_LOG_SIZE` environment variables.

The `system_n.log` file can contain the following information:

- Agent version information, written when the agent instance starts up.
- Logs for the agent background processes.
- WebSocket connection errors.
- Cache stats and removal of old POST data preservation files.
- Agent notifications.

The default value of the `AM_SYSTEM_LOG_LEVEL` variable is `Error`. Increase it to `Message` or `All` for fine-grained detail.

Valid values for the variable are:

- All
- Message
- Warning
- Error

- Info

***`AM_SYSTEM_LOG_PATH`***

The directory where the `system_n.log` file is stored.

By default, this environment variable is configured to `/path/to/web_agents/agent_type/log`.

***`AM_SYSTEM_LOG_SIZE`***

Thesize, in bytes, of the `/path/to/web_agents/agent_type/log/system_n.log` file.

The default is `0`, which specifies that the log file size is unlimited. Valid ranges for this variable go from 0 to 4294967295 bytes (4GB).

***`AM_SYSTEM_PIPE_DIR`***

(Unix only) Thedirectory where the agent instances will store their temporary pipe files.

The default is `/path/to/web_agents/agent_type/log/`.

## *Web Agent Installer Environment Variables*

Use the following properties during agent installation.

For example, use the `AM_SSL_*` environment properties to install agents when SSL is already configured in AM or in the agent container.

The install process will set the values in the environment variables as part of

the agent configuration if the process is successful. You may also use these settings with the **`agentadmin -V[i]`** command if you want to test values different from those already configured for the agent instance.

***`AM_PROXY_HOST`***

When AM and the agent communicate through a proxy configured in forward proxy mode, this environment variable specifies the proxy FQDN.

***`AM_PROXY_PASSWORD`***

When AM and the agent communicate through a proxy configured in forward proxy mode, and the proxy requires that the agent authenticates using Basic Authentication, this environment variable specifies the password of the agent.

***`AM_PROXY_USER`***

When AM and the agent communicate through a proxy configured in forward proxy mode, and the proxy requires the agent authenticates using Basic Authentication, this environment variable specifies the user name of the agent.

***`AM_PROXY_PORT`***

When AM and the agent communicate through a proxy configured in forward proxy mode, this variable specifies the proxy port number.

***APACHE_RUN_USER***

The user running the Apache HTTP or IBM HTTP Server.

Use this variable if there is no Apache user defined in the `httpd.conf` file.

***APACHE_RUN_GROUP***

The group to which the user running the Apache HTTP or IBM HTTP Server belongs.

Use this variable if there is no Apache group defined in the `httpd.conf` file.

***AM_SSL_SCHANNEL***

(Windows only) Specifies whether the agent installation process should use the Windows Secure Channel API. Possible values are:

- `0`. Disable Windows Secure Channel API support. The agent will use OpenSSL libraries instead.

  Ensure that the OpenSSL libraries are in the appropriate place, as specified in the OpenSSL Library Location by Operating System table.
- `1`. Enable Windows Secure Channel API support.

***AM_SSL_KEY***

(OpenSSL only) When AM is configured to perform client authentication, this environment variable specifies a PEM file that contains the private key corresponding to the certificate specified in the `AM_SSL_CERT` environment variable. For example, `/opt/certificates/client-private-key.pem` or `C:\Certificates\client-private-key.pem`.

***AM_SSL_PASSWORD***

(OpenSSL only) When AM is configured to perform client authentication, this environment variable specifies the obfuscated password of the private key configured in the `AM_SSL_KEY` variable. Configure this variable only if the private key is password-protected.

To obfuscate the password, use the **`agentadmin --p`** command. For example:

1. Unix
2. Windows

```
$ /path/to/web_agents/agent_type/bin/> agentadmin --p
"Encryption Key" “cat certificate_password.file”
Encrypted password value:
zck+6RKqjtc=com.forgerock.agents.config.cert.key.password =
zck+6RKqjtc=
C:\path\to\web_agents\agent_type\bin> agentadmin.exe --p
"Encryption_Key" "Certificate_File_Password"
Encrypted password value: zck+6RKqjtc=
```

***AM_SSL_CIPHERS***

(OpenSSL only) The list of ciphers to support. The list consists of one or more cipher strings separated by colons, as defined in the man page for ciphers available at http://www.openssl.org/docs/apps/ciphers.htm.

For example, `HIGH:MEDIUM`.

***AM_SSL_CERT***

When AM is configured to perform client authentication, this environment variable specifies a PEM file that contains the certificate chain for the agent.

For example, `/opt/certificates/client-cert.pem`, `C:\Certificates\client-cert.pem` (Windows with OpenSSL), or `Cert:\LocalMachine\My location` (Windows with the Windows Secure Channel API).

***AM_SSL_CA***

When configuring the agent to validate AM's certificate, this environment variable specifies a PEM file that contains the certificates required to validate AM's server certificate. For example, `/opt/certificates/ca.pem`, `C:\Certificates\ca.pem` (Windows with OpenSSL), or `Cert:\LocalMachine\Ca` (Windows with the Windows Secure Channel API).

## agentadmin Command

The **`agentadmin`** command manages Web Agent installation. It returns `EXIT_SUCCESS` (or `0`) when it completes successfully, and `EXIT_FAILURE` (or a code greater than zero) when it fails.

The following options are supported:

***--i***

Install a new agent instance.

Usage: **`agentadmin --i`**

***--s***

Silently, non-interactively, install a new agent instance.

Usage: **`agentadmin --s`** ***`web-server-config-file openam-url agent-url realm agent-profile-name agent-profile-password`*** **`[--changeOwner] [--acceptLicense] [--forceInstall]`**

*web-server-config-file*

(Apache HTTP Server) The full path to the Apache HTTP server configuration file. The installer modifies this file to include the agent configuration and module.

(Microsoft IIS) The ID number of the IIS site in which to install the web agent. To list the available sites in an IIS server and the relevant ID numbers, run **`agentadmin.exe --n`**.

*openam-url*

The full URL of the AM instance that the agent will use. Ensure the deployment URI is specified.

Example: `https://openam.example.com:8443/openam`

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

*agent-url*

Enter the full URL of the server on which the agent is running.

Example: `http://www.example.com:80`

*realm*

Enter the AM realm containing the agent profile.

*agent-profile-name*

The name of the agent profile in AM.

*agent-profile-password*

The full path to the agent profile password file.

***`--changeOwner`***

(Apache web agent for Unix only) Use this option to set the ownership of created directories to the user and group as specified in the Apache configuration file.

Note that this option will only change the ownership of the files and directories if you run the **`agentadmin`** command as the `root` user or using the **`sudo`** command.

If you cannot run the **`agentadmin`** as the `root` user or using the **`sudo`** command, you must change the ownership manually.

***`--acceptLicense`***

When you run certain commands, you will be prompted to read and accept the software license agreement. You can suppress the license agreement prompt by including the optional `--acceptLicence` parameter. Specifying this options indicates that you have read and accepted the terms stated in the license.

To view the license agreement, open `/path/to/web_agents/agent_type/legal/Forgerock_License.txt` .

***--forceInstall***

Add this option to proceed with a silent installation even if it cannot connect to the specified AM server during installation, rather than exiting.

***--n***

(IIS web agent only) List the sites available in an IIS server.

Example:

```
c:\web_agents\iis_agent\bin> agentadmin.exe --nIIS Server Site
configuration:
     ====================================
     id       details
     ====================================

     Default Web Site
     application path:/, pool DefaultAppPool
     1.1.1    virtualDirectory path:/, configuration:
C:\inetpub\wwwroot\web.config

     MySite
     application path:/, pool: MySite
     2.1.1    virtualDirectory path:/, configuration
C:\inetpub\MySite\web.config
     application path:/MyApp1, pool: MySite
```

***--l***

List existing configured agent instances.

Usage: **`agentadmin --l`**

Example:

```
$ ./agentadmin --l
OpenAM Web Agent configuration instances:

      id:            agent_1
      configuration:
/opt/web_agents/apache24_agent/bin/../instances/agent_1
      server/site:   /etc/httpd/conf/httpd.conf
      id:            agent_2
      configuration:
/opt/web_agents/apache24_agent/bin/../instances/agent_2
      server/site:   /etc/httpd/conf/httpd.conf

      id:            agent_3
      configuration:
/opt/web_agents/apache24_agent/bin/../instances/agent_3
      server/site:   /etc/httpd/conf/httpd.conf
```

***--g***

(IIS web agent only) Remove all web agent instances and libraries from an IIS installation.

Usage: **`agentadmin.exe --g`**

For more information, see To Remove Web Agents from IIS.

***--e***

(IIS web agent only) Enable an existing agent instance.

Usage: **`agentadmin.exe --e`** ***`agent-instance`***

For more information, see To Disable and Enable Web Agents.

***--d***

(IIS web agent only) Disable an existing agent instance.

Usage: **`agentadmin.exe --d`** ***`agent-instance`***

For more information, see To Disable and Enable Web Agents.

***--o***

(IIS web agent only) Modify Access Control Lists (ACLs) for files and folders related to a web agent instance.

Usage: **`agentadmin.exe --o`** "***`identity_or_siteID`***" "***`directory`***" **`[--siteId]`**

Usage: **`agentadmin.exe --o`** "***`directory`***" **`--addAll --removeAll`**

*"identity_or_siteID"*

Specifies the identity to be added to the directory's ACLs. When used with the `--siteId` option, it specifies an IIS site ID.

*"directory"*

Specifies the directory that would be modified.

***`[--siteId]`***

Specifies that the **agentadmin** should use `identity_or_siteID` as an IIS site ID.

***--addAll***
Add all IIS application pool identities to the directory's ACLs. This option is not compatible with the `--removeAll` option.

***--removeAll***
Remove all IIS application pool identities from the directory's ACLs. This option is not compatible with the `--addAll` option.

Examples:

```
C:\web_agents\iis_agent\bin> agentadmin.exe --o "IIS_user1"
"C:\web_agents\iis_agent\lib"
```

```
C:\web_agents\iis_agent\bin> agentadmin.exe --o "2"
"C:\web_agents\iis_agent\lib" --siteId
```

```
C:\web_agents\iis_agent\bin> agentadmin.exe --o
"C:\web_agents\iis_agent\lib" --addAll
```

***--r***
Remove an existing agent instance.

Usage: **agentadmin --r *agent-instance***

*agent-instance*
The ID of the web agent configuration instance to remove.

Respond `yes` when prompted to confirm removal.

On IIS web agents, the `--r` option does not remove the web agent libraries since they can be in use by other web agent instances configured on the same site. To remove all web agent instances and libraries, use the `--g` option instead.

***--k***
Generate a new signing key.

Usage: **agentadmin --k**

Examples:

1. Unix
2. Windows

```
$ cd /web_agents/apache24_agent/bin/$ ./agentadmin --k
Encryption key value: YWM0OThlMTQtMzMxOS05Nw==
```

```
C:\> cd web_agents\apache24_agent\bin
C:\web_agents\apache24_agent\bin> agentadmin --k
Encryption key value: YWM0OThlMTQtMzMxOS05Nw==
```

***--p***

Use a generated encryption key to encrypt a new password.

Usage: **`agentadmin --p`** ***`encryption-key password`***

*encryption-key*

An encryption key, generated by the **`agentadmin --k`** command.

*password*

The password to encrypt.

Examples:

1. Unix
2. Windows

```
$ ./agentadmin --p "YWM0OThlMTQtMzMxOS05Nw==" "cat
newpassword.file"
Encrypted password value: 07bJOSeM/G8ydO4=
```

```
C:\path\to\web_agents\apache24_agent\bin> agentadmin.exe --p
"YWM0OThlMTQtMzMxOS05Nw==" "newpassword"
Encrypted password value: 07bJOSeM/G8ydO4=
```

***--V[i]***

Validate an agent instance.

Usage:

```
agentadmin --V[i] agent_instance [user name] [password file] [realm]
```

*[i]*

(Optional) Ensures that the core init and shutdown agent sequences are working as expected.

*Do not use this option while the agent is actively protecting a website*. Doing so may make the agent instance unresponsive, causing unexpected service outages.

*agent_instance*

(Required) The agent instance where to run the validation tests. For example, `agent_1`.

***user name***

(Optional) A user ID that exists in the AM server. Required only for the `validate_session_profile` test. For example, `demo`.

***password file***

(Optional) A file containing the password of the user ID used for the `validate_session_profile` test. For example, `/tmp/passwd.txt`

***realm***

(Optional) The realm of the user ID used for the `validate_session_profile` test. For example, `/customers`

Validation mode performs tests to ensure the following points:

- The agent can reach the AM server(s) configured in AM Connection URL.
- Critical bootstrap properties are set. For more information, see Configuration Location.
- SSL libraries are available and that SSL configuration properties are set, if the agent is configured for SSL communication.
- The agent can log in to AM to fetch the agent profile.
- The system has enough RAM and shared memory.
- The agent can log in to AM with the provided user and password credentials.
- WebSocket connections are available between the agent and AM.
- The core init and shutdown agent sequences are working as expected.

  NOTE

  This validation requires the `--Vi` flag. *Do not use this option while the agent instance is actively protecting a website*. Doing so may make the agent instance unresponsive, causing unexpected service outages.

- (IIS agent only) That IIS is configured for running application pools in Integrated mode.

IMPORTANT

IMPORTANT

On Unix, run the **agentadmin --V[i]** validator command as the same user that runs the web server.

For example, to use the Apache HTTP Server `daemon` user:

```
$ sudo -u daemon ./bin/agentadmin --V agent_1
```

Running the command as a different user may cause the `log/system_0.log` and `log/monitor_0.pipe` files to be created with permissions that prevent the agent from writing to them. In this case, you may see an error such as:

```
2018-09-19 13:54:52 GMT ERROR   [0x7f0c9cf05700:22420]: unable
to open event channel
```

Make sure the user running the command has execute permission on the following directories:

- `/web_agents/apache24_agent/instances/agent_nnn`
- `/web_agents/apache24_agent/log`

Example:

```
$ ./agentadmin --Vi agent_1 demo passwd.txt /
Saving output to
/web_agents/apache24_agent/bin//../log/validate_20180831121402.
log

Running configuration validation for agent_1:

Agent instance is configured with 1 naming.url value(s):
1. https://openam.example.com:8443/openam is valid
selected https://openam.example.com:8443/openam as naming.url
value
validate_bootstrap_configuration: ok
validate_ssl_libraries: ok
validate_agent_login: ok
get_allocator_blockspace_sz(): trying for configured cache size
16777216 bytes
validate_system_resources: ok
validate_session_profile: ok
validate_websocket_connection: ok
validate_worker_init_shutdown: ok
```

```
Result: 7 out of 7 tests passed, 0 skipped.
```

***--v***

Display information about **agentadmin** build and version numbers, and available system resources.

For example:

```
AM Web Agent for IIS Server 7.5, 8.x
Version: 5.9.1
Revision: ab12cde
Build machine: WIN-6R2CH15R77
Build date: Nov  8 2016 11:30:18

System Resources:
total memory size: 7.7GB
pre-allocated session/policy cache size: 1.0GB
log buffer size: 128.5MB
min audit log buffer size: 2MB, max 2.0GB
total disk size: 162.4GB
free disk space size: 89.6GB

System contains sufficient resources (with remote audit log
feature enabled).
```

## Configuring Apache HTTP Server as a Reverse Proxy Example

This section contains an example configuration of Apache as a reverse proxy between AM and Web Agent. You can use any reverse proxy that supports the WebSocket protocol.

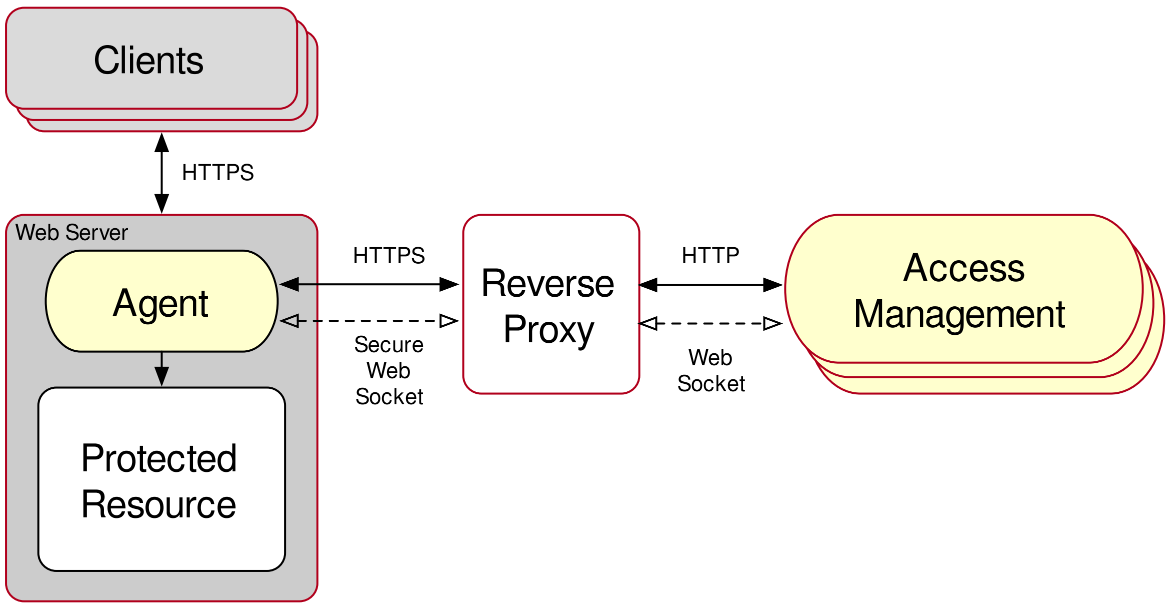


*Figure 6. Reverse Proxy Configured Between the Agent and AM*

For information about how to configure Apache for load balancing, and other requirements for your environment, see the Apache documentation.

1. Locate the `httpd.conf` file in your deployed reverse proxy instance.
2. Add the modules required for a proxy configuration, as follows:

```
# Modules required for proxy
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
LoadModule proxy_wstunnel_module
modules/mod_proxy_wstunnel.so
```

   The `mod_proxy_wstunnel.so` module is required to support the WebSocket protocol used for communication between AM and the agents.

3. Add the proxy configuration inside the `VirtualHost` context. Consider the following directives:

```
<VirtualHost 192.168.1.1>
...
# Proxy Config
RequestHeader set X-Forwarded-Proto "https" (1)
ProxyPass "/openam/notifications"
"ws://openam.example.com:8080/openam/notifications"
Upgrade=websocket (2)
ProxyPass "/openam"
"http://openam.example.com:8080/openam" (3)
ProxyPassReverseCookieDomain "openam.internal.example.com"
"proxy.example.com" (4)
ProxyPassReverse "/openam"
"http://openam.example.com:8080/openam" (5)
...
</VirtualHost>
```

   (1) RequestHeader: Set to `https` or `http`, depending on the proxy configuration. If the proxy is configured for https, as in the above example, set to `https`. Otherwise, set `http`. In a later step, you configure AM to recognize the forwarded header and use it in the `goto` parameter for redirecting back to the agent after authentication.

   (2) ProxyPass: Set to allow WebSocket traffic between AM and the agent. If HTTPS is configured between the proxy and AM, set to use the `wss` protocol instead of

`ws`.

(3) ProxyPass: Set to allow HTTP traffic between AM and the agent.

(4) ProxyPassReverseCookieDomain: Set to rewrite the domain string in `Set-Cookie`headers in the format internal domain (AM's domain) public domain (proxy's domain).

(5) ProxyPassReverse: Set to the same value configured for the `ProxyPass` directive.

For more information about configuring Apache as a reverse proxy, see the Apache documentation.

4. Restart the reverse proxy instance.
5. Configure AM to recover the forwarded header you configured in the reverse proxy. Also, review other configurations that may be required in an environment that uses reverse proxies. For more information, see Regarding Communication Between AM and Agents

# 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 group***
A group of agent instances that share runtime resources and shared memory.

***Application***

In general terms, a service exposing protected resources.

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

***Application type***
Application types act as templates for creating policy applications.

Application types define a preset list of actions and functional logic, such as policy lookup and resource comparator logic.

Application types also define the internal normalization, indexing logic, and comparator logic for applications.

***Attribute-based access control (ABAC)***
Access control that is based on attributes of a user, such as how old a user is or whether the user is a paying customer.

***Authentication***
The act of confirming the identity of a principal.

***Authentication chaining***
A series of authentication modules configured together which a principal must negotiate as configured in order to authenticate successfully.

***Authentication level***
Positive integer associated with an authentication module, usually used to require success with more stringent authentication measures when requesting resources requiring special protection.

***Authentication module***
AM authentication unit that handles one way of obtaining and verifying credentials.

***Authentication Session***
The interval while the user or entity is authenticating to AM.

***Session***
The interval that starts after the user has authenticated and ends when the user logs out, or when their session is terminated. For browser-based clients, AM manages user sessions across one or more applications by setting a session cookie.

***Authorization***
The act of determining whether to grant or to deny a principal access to a resource.

***Authorization Server***
In OAuth 2.0, issues access tokens to the client after authenticating a resource owner and confirming that the owner authorizes the client to access the protected resource. AM can play this role in the OAuth 2.0 authorization framework.

***Auto-federation***
Arrangement to federate a principal's identity automatically based on a common attribute value shared across the principal's profiles at different providers.

***Bulk federation***
Batch job permanently federating user profiles between a service provider and an identity provider, based on a list of matched user identifiers that exist on both providers.

***Centralized configuration mode***
AM stores the agent properties in the AM configuration store. See also local configuration mode.

The configuration mode is defined by Location of Agent Configuration Repository.

***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 applications involved in SSO when a session ends.

***Delegation***
: Granting users administrative privileges with AM.

***Entitlement***
: Decision that defines which resource names can and cannot be accessed for a given identity in the context of a particular application, which actions are allowed and which are denied, and any related advice and attributes.

***Extended metadata***
: Federation configuration information specific to AM.

***Extensible Access Control Markup Language (XACML)***
: Standard, XML-based access control policy language, including a processing model for making authorization decisions based on policies.

***Federation***
: Standardized means for aggregating identities, sharing authentication and authorization data information between trusted providers, and allowing principals to access services across different providers without authenticating repeatedly.

***Fedlet***
: Service provider application capable of participating in a circle of trust and allowing federation without installing all of AM on the service provider side; AM lets you create Java Fedlets.

***Hot swappable***
: Refers to configuration properties for which changes can take effect without restarting the container where AM runs.

***Identity***
: Set of data that uniquely describes a person or a thing such as a device or an application.

***Identity federation***

Linking of a principal’s identity across multiple providers.

***Identity provider (IdP)***

Entity that produces assertions about a principal (such as how and when a principal authenticated, or that the principal’s profile has a specified attribute value).

***Identity repository***

Data store holding user profiles and group information; different identity repositories can be defined for different realms.

***Identity store***

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

***Java agent***

Java web application installed in a web container that acts as a policy enforcement point, filtering requests to other applications in the container with policies based on application resource URLs.

***Local configuration mode***

The Web Agent installer creates the file `/web_agents/agent_type/instances/agent_nnn/config/agent.conf` to store the agent configuration properties. See also centralized configuration mode.

The configuration mode is defined by Location of Agent Configuration Repository.

***Metadata***

Federation configuration information for a provider.

***Policy***

Set of rules that define who is granted access to a protected resource when, how, and under what conditions.

***Policy agent***

Java, web, or custom agent that intercepts requests for resources, directs principals to AM for authentication, and enforces policy decisions from AM.

***Policy Administration Point (PAP)***

Entity that manages and stores policy definitions.

***Policy Decision Point***

Entity that evaluates access rights, and then issues authorization decisions.

***Policy Enforcement Point (PEP)***

Entity that intercepts a request for a resource, and then enforces policy decisions from a policy decision point.

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

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

***Principal***

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

When an AM identity successfully authenticates, AM associates the identity with the Principal.

***Privilege***

In the context of delegated administration, a set of administrative tasks that can be performed by specified identities in a given realm.

***Provider federation***

Agreement among providers to participate in a circle of trust.

***Realm***

AM unit for organizing configuration and identity information.

Realms can be used for example when different parts of an organization have different applications and identity stores, and when different organizations use the same AM deployment.

Administrators can delegate realm administration. The administrator assigns administrative privileges to users, allowing them to perform administrative tasks within the realm.

***Resource***

Something a user can access over the network such as a web page.

Defined as part of policies, these can include wildcards in order to match multiple actual resources.

***Resource owner***

In OAuth 2.0, entity who can authorize access to protected web resources, such as an end user.

***Resource server***

In OAuth 2.0, server hosting protected web resources, capable of handling access tokens to respond to requests for such resources.

***Response attributes***

Defined as part of policies, these allow AM to return additional information in the form of "attributes" with the response to a policy decision.

***Role based access control (RBAC)***

Access control that is based on whether a user has been granted a set of permissions (a role).

***Security Assertion Markup Language (SAML)***

Standard, XML-based language for exchanging authentication and authorization data between identity providers and service providers.

***Service provider (SP)***

Entity that consumes assertions about a principal (and provides a service that the principal is trying to access).

***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 CTS-based sessions, the session token is used to track a principal’s session.

***Single log out (SLO)***

Capability allowing a principal to end a session once, thereby ending her session across multiple applications.

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

Capability allowing a principal to authenticate once and gain access to multiple applications without authenticating again.

***Site***

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

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

***Standard metadata***

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

***Stateless Service***

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

All AM services are stateless unless otherwise specified.

***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.

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