



# Administration Guide

OpenAM 11

Mark Craig  
Vanessa Richie  
Mike Jang

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

---

Copyright © 2011-2017 ForgeRock AS.

## Abstract

Guide to configuring and using OpenAM features. OpenAM provides open source Authentication, Authorization, Entitlement and Federation software.



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

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

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

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

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

DejaVu Fonts

Bitstream Vera Fonts Copyright

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

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

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

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

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

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

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

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

Arev Fonts Copyright

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

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

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

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

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

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

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

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

FontAwesome Copyright

Copyright (c) 2017 by Dave Gandy, <http://fontawesome.io>.

This Font Software is licensed under the SIL Open Font License, Version 1.1. This license is available with a FAQ at: <http://scripts.sil.org/OFL>.

---

# Table of Contents

Preface .....	vi
1. Who Should Use this Guide .....	vi
2. Formatting Conventions .....	vi
3. Accessing Documentation Online .....	vii
4. Using the ForgeRock.org Site .....	vii
1. Administration Interfaces & Tools .....	1
1.1. OpenAM Web-Based Console .....	1
1.2. OpenAM Command-Line Tools .....	3
1.3. OpenAM ssoadm.jsp .....	4
2. Defining Authentication Services .....	5
2.1. About Authentication in OpenAM .....	5
2.2. Configuring Authentication Modules .....	7
2.3. Configuring Authentication Chains .....	60
2.4. Post Authentication Plugins .....	62
2.5. Authenticating To OpenAM .....	63
2.6. Configuring Account Lockout .....	66
2.7. Configuring Session Quotas .....	67
3. Defining Authorization Policies .....	69
3.1. About Authorization in OpenAM .....	69
3.2. Configuring Policies .....	70
3.3. How OpenAM Reaches Policy Decisions .....	74
3.4. Managing Policies Outside the Console .....	75
3.5. Delegating Policy Management & Decisions .....	77
4. Defining Entitlements .....	78
4.1. About Entitlements .....	78
4.2. Managing Entitlements on the Command Line .....	79
5. Configuring Realms .....	81
6. Configuring Policy Agent Profiles .....	86
6.1. Identity Gateway or Policy Agent? .....	86
6.2. Kinds of Agent Profiles .....	87
6.3. Creating Agent Profiles .....	87
6.4. Delegating Agent Profile Creation .....	90
6.5. Configuring Web Policy Agents .....	90
6.6. Configuring J2EE Policy Agents .....	113
6.7. Configuring Web Service Provider Policy Agents .....	139
6.8. Configuring Web Service Client Policy Agents .....	142
6.9. Configuring Security Token Service Client Policy Agents .....	145
6.10. Configuring Version 2.2 Policy Agents .....	149
6.11. Configuring OAuth 2.0 & OpenID Connect 1.0 Clients .....	149
6.12. Configuring Agent Authenticators .....	151
7. Working with Mobile Devices & Applications .....	153
7.1. Simplifying Access on Mobile Devices .....	153
7.2. Protecting Access for Mobile Users .....	156
7.3. Simplifying Access with REST APIs .....	157

7.4. Getting Source Code for Sample Mobile Applications .....	158
8. Configuring Password Reset .....	160
8.1. About Password Reset .....	160
8.2. Resetting Forgotten Passwords .....	160
9. Configuring Single Sign-On within One Domain .....	168
9.1. The Basics of the HTTP Cookie .....	168
9.2. Cookies and the SSO Session Process .....	169
9.3. Potential Problems .....	170
9.4. Configure SSO on One Domain .....	171
10. Configuring Cross-Domain Single Sign On .....	172
11. Configuring the Dashboard Service .....	179
11.1. About the Dashboard Service .....	179
11.2. Setting Up the Dashboard Service .....	180
11.3. Configuring Dashboard Service for a Realm .....	180
11.4. Adding Applications to a User's Dashboard .....	181
12. Managing SAML 2.0 Federation .....	182
12.1. About SAML 2.0 SSO & Federation .....	182
12.2. Setting Up SAML 2.0 SSO .....	183
12.3. Configuring Identity Providers .....	191
12.4. Configuring Service Providers .....	196
12.5. Configuring Circles of Trust .....	201
12.6. Configuring Google Apps as a Remote Service Provider .....	202
12.7. Configuring Salesforce CRM as a Remote Service Provider .....	203
12.8. Using SAML 2.0 Single Sign-On & Single Logout .....	206
12.9. Configuring OpenAM For the ECP Profile .....	212
12.10. Managing Federated Accounts .....	213
12.11. Using SAML 2.0 Artifacts .....	221
13. Managing OAuth 2.0 Authorization .....	222
13.1. About OAuth 2.0 Support in OpenAM .....	222
13.2. Configuring the OAuth 2.0 Authorization Service .....	229
13.3. Registering OAuth 2.0 Clients With the Authorization Service .....	230
13.4. Managing OAuth 2.0 Tokens .....	231
13.5. Configuring OpenAM as Authorization Server & Client .....	232
13.6. Configuring OpenAM as Both SAML 2.0 Service Provider & OAuth 2.0 Authorization Server .....	236
13.7. Security Considerations .....	239
14. Managing OpenID Connect 1.0 Authorization .....	240
14.1. About OpenID Connect 1.0 Support in OpenAM .....	240
14.2. Configuring OpenAM As an OpenID Connect Provider .....	243
14.3. Configuring OpenAM For OpenID Connect Discovery .....	243
14.4. Registering OpenID Connect Clients .....	245
14.5. Managing User Sessions .....	246
14.6. Client Examples .....	247
14.7. Security Considerations .....	251
15. Backing Up and Restoring OpenAM Configurations .....	252
16. Managing Certificates .....	254
17. Monitoring OpenAM Services .....	260

17.1. Monitoring Interfaces .....	260
17.2. Is OpenAM Running? .....	263
17.3. Log Management .....	263
17.4. Session Management .....	265
18. Tuning OpenAM .....	266
18.1. OpenAM Server Settings .....	266
18.2. Java Virtual Machine Settings .....	268
18.3. Caching in OpenAM .....	270
19. Changing Host Names .....	275
20. Securing OpenAM .....	278
20.1. Avoiding Obvious Defaults .....	278
20.2. Protecting Network Access .....	278
20.3. Securing OpenAM Administration .....	280
20.4. Securing Communications .....	281
20.5. Administering the amadmin Account .....	281
21. Troubleshooting .....	284
OpenAM Glossary .....	288
A. Release Levels & Interface Stability .....	293
A.1. ForgeRock Product Release Levels .....	293
A.2. ForgeRock Product Interface Stability .....	294
Index .....	295

# Preface

This guide shows you how to configure, maintain, and troubleshoot OpenAM for single sign on and authorization, password reset, account lockout, cross-domain single sign on, and federation.

## 1. Who Should Use this Guide

This guide is written for access management designers and administrators who build, deploy, and maintain OpenAM services for their organizations. This guide covers the tasks you might repeat throughout the life cycle of an OpenAM release used in your organization.

This guide starts by introducing the OpenAM administrative interfaces and tools, and by showing how to manage OpenAM services. This guide continues by showing how to configure the principle features of OpenAM. It then demonstrates how to backup, restore, monitor, tune, and troubleshoot, OpenAM services.

You do not need to be an OpenAM wizard to learn something from this guide, though a background in access management and maintaining web application software can help. You do need some background in managing services on your operating systems and in your application servers. You can nevertheless get started with this guide, and then learn more as you go along.

## 2. Formatting Conventions

Most examples in the documentation are created in GNU/Linux or Mac OS X operating environments. If distinctions are necessary between operating environments, examples are labeled with the operating environment name in parentheses. To avoid repetition file system directory names are often given only in UNIX format as in `/path/to/server`, even if the text applies to `C:\path\to\server` as well.

Absolute path names usually begin with the placeholder `/path/to/`. This path might translate to `/opt/`, `C:\Program Files\`, or somewhere else on your system.

Command-line, terminal sessions are formatted as follows:

```
$ echo $JAVA_HOME
/path/to/jdk
```

Command output is sometimes formatted for narrower, more readable output even though formatting parameters are not shown in the command.

Program listings are formatted as follows:

```
class Test {  
    public static void main(String [] args) {  
        System.out.println("This is a program listing.");  
    }  
}
```

## 3. Accessing Documentation Online

ForgeRock publishes comprehensive documentation online:

- The [ForgeRock Knowledge Base](#) offers a large and increasing number of up-to-date, practical articles that help you deploy and manage ForgeRock software.

While many articles are visible to community members, ForgeRock customers have access to much more, including advanced information for customers using ForgeRock software in a mission-critical capacity.

- ForgeRock product documentation, such as this document, aims to be technically accurate and complete with respect to the software documented. It is visible to everyone and covers all product features and examples of how to use them.

## 4. Using the ForgeRock.org Site

The [ForgeRock.org](#) site has links to source code for ForgeRock open source software, as well as links to the ForgeRock forums and technical blogs.

If you are a *ForgeRock customer*, raise a support ticket instead of using the forums. ForgeRock support professionals will get in touch to help you.

## Chapter 1

# Administration Interfaces & Tools

This chapter provides a brief introduction to the web-based OpenAM console. It also lists and describes each command line interface (CLI) administration tool.

## 1.1. OpenAM Web-Based Console

After you install OpenAM, login to the web-based console as OpenAM Administrator, `amadmin` with the password you set during installation. Navigate to a URL such as `http://openam.example.com:8080/openam`. In this case, communications proceed over the HTTP protocol to a FQDN (`openam.example.com`), over a standard Java EE web container port number (8080), to a specific deployment URI (`/openam`).



The screenshot shows the OpenAM Web-Based Console interface. At the top, there is a navigation menu with the following tabs: **Common Tasks**, **Access Control**, **Federation**, **Configuration**, and **Sessions**. Below the menu, the **Configuration** section is active, displaying a list of tasks:

- Create SAMLv2 Providers**: Use these work flows to create hosted or remote identity and service providers for SAMLv2 Federation.
  - Create Hosted Identity Provider
  - Create Hosted Service Provider
  - Register Remote Identity Provider
  - Register Remote Service Provider
- Configure OAuth2**: This task configures OAuth2 per realm. Each realm can act as an authorization server.
  - Configure OAuth2
- Create Fedlet**: Create a Fedlet to enable federation between an identity provider hosted on this instance of OpenAM and a remote service provider that does not have a federation solution. Before beginning, a hosted identity provider must be configured.
  - Create Fedlet
- Configure Google Apps**: Integrate OpenAM with Google Apps web applications to create a single sign-on environment. Before beginning, a hosted identity provider and Circle of Trust must be configured.
  - Configure Google Apps
- Configure Salesforce CRM**: Integrate OpenAM with Salesforce CRM to create a single sign-on environment. Before beginning, a SAMLv2 hosted identity provider and Circle of Trust must be configured.
  - Configure Salesforce CRM
- Test Federation Connectivity**: Use this automated test to determine if federation connections are being made successfully, or to identify where issues might be located.
  - Test Federation Connectivity
- Get Product Documentation**: Launch the OpenAM product documentation page.
  - Get Product Documentation

When you login as the OpenAM Administrator, `amadmin`, you have access to the complete OpenAM console. In addition, OpenAM has set a cookie in your browser that lasts until the session expires, you logout, or you close your browser.<sup>1</sup>

When you login to the OpenAM console as a non-administrative end user, you do not have access to the administrative console. Your access is limited to a configuration page with your account information.

**Babs Jensen**

\* Indicates required field

First Name:

\* Last Name:

\* Full Name:

Password: [Edit](#)

Email Address:

Telephone Number:

Home Address:

Password Reset Options: [Edit](#)

Universal ID: `id=bjensen,ou=user,dc=openam,dc=forgerock,dc=org`

If you configure OpenAM to grant administrative capabilities to another user, then that user also sees the console after login. For instance, the OpenAM Administrator granted Kirsten Vaughan privileges to administer the OpenAM Top Level Realm. (This can be done through the console under Access Control > / (Top Level Realm) > Privileges. Kirsten has authorization to read and write policy properties and configured policy agent properties.) When Kirsten logs in, she sees only part of the console capabilities.<sup>2</sup>

**Access Control**

A realm is the unit that OpenAM uses to organize configuration information. Authentication properties, authorization policies, data stores, subjects and other data can be defined within the realm. The top level realm is created when you deploy OpenAM. The top level realm is the root of the OpenAM instance and contains OpenAM configuration data.

**Realms**

Realms (1 Item(s))	
Realm Name	Location
<a href="#">/(Top Level Realm)</a>	/

<sup>1</sup>Persistent cookies can remain valid when you close your browser. This section reflects OpenAM default behavior before you configure additional functionality.

<sup>2</sup>For more on delegated administration, see the chapter covering realms.

## 1.2. OpenAM Command-Line Tools

The script tools in the following list have `.bat` versions for use on Microsoft Windows.

You can install the following OpenAM command-line tools.

### **agentadmin**

This tool lets you manage OpenAM policy agent installations.

Unpack this tool as part of policy agent installation.

### **ampassword**

This tool lets you change OpenAM Administrator passwords, and display encrypted password values.

Install this from the `SSOAdminTools-11.0.0.zip`.

### **amverifyarchive**

This tool checks log archives for tampering.

Install this from `SSOAdminTools-11.0.0.zip`.

### **openam-distribution-configurator-11.0.0.jar**

This executable `.jar` file lets you perform a silent installation of an OpenAM server with a configuration file. For example, the `java -jar configurator.jar -f config.file` command couples the `configurator.jar` archive with the `config.file`. The `sampleconfiguration` file provided with the tool is set up with the format for the `config.file`, and it must be adapted for your environment.

Install this from `SSOConfiguratorTools-11.0.0.zip`.

### **ssoadm**

This tool provides a rich command-line interface for the configuration of OpenAM core services.

In a test environment you can activate `ssoadm.jsp` to access the same functionality in your browser. Once active, you can use many features of the `ssoadm` command, by navigating to the `ssoadm.jsp` URI, in a URL such as `http://openam.example.com:8080/openam/ssoadm.jsp`.

Install this from `SSOAdminTools-11.0.0.zip`.

To translate settings applied in OpenAM console to service attributes for use with `ssoadm`, login to the OpenAM console as `amadmin` and access the services page, in a URL such as `http://openam.example.com:8080/openam/services.jsp`.

The commands access the OpenAM configuration over HTTP (or HTTPS). When using the administration commands in a site configuration, the commands access the configuration through the front end load balancer.

Sometimes a command cannot access the load balancer, because:

- Network routing restrictions prevent the tool from accessing the load balancer.
- For testing purposes, the load balancer uses a self-signed certificate for HTTPS, and the tool does not have a way of trusting the self-signed certificate.
- The load balancer is temporarily unavailable.

In such cases you can work around the problem by adding an option such as the following to the **java** command in the tool's script. The option sets a comma-separated list of key-value pairs, where the key is the load balancer URL and the value is the server URL. (This all belongs on one line with no spaces in the script.)

```
-D"com.iplanet.am.naming.map.site.to.server=https://lb.example.com:443/openam=  
http://server1.example.com:8080/openam,https://lb.example.com:443/openam=  
http://server2.example.com:8080/openam"
```

In the above example the load balancer is on the **lb** host, <https://lb.example.com:443/openam> is the site name, and the OpenAM servers in the site are on **server1** and **server2**.

## 1.3. OpenAM ssoadm.jsp

You can use the **ssoadm.jsp** page to access a large subset of the configuration capabilities of the **ssoadm** command. Yet, **ssoadm.jsp** is disabled by default to prevent potential misuse.

### *Procedure 1.1. To Enable ssoadm.jsp*

1. Login as OpenAM administrator, **amadmin**.
2. Click Configuration > Servers and Sites > Servers > *URL of your server*.
3. Click Advanced to display the Advanced Properties table, and then click Add. In the text boxes that appear, include the following information, and then click Save.

#### **Property Name**

ssoadm.disabled

#### **Property Value**

false

4. To see if the change worked, navigate to the URL of OpenAM with the **/ssoadm.jsp** URI. For the aforementioned URL, you would navigate to <http://openam.example.com:8080/openam/ssoadm.jsp>.

## Chapter 2

# Defining Authentication Services

An *authentication* service confirms the identity of a user or a client application.

This chapter describes how to configure authentication in OpenAM.

## 2.1. About Authentication in OpenAM

Access management is about controlling access to resources. OpenAM plays a role similar to border control at an international airport. Instead of having each and every airline company deal with access to each destination, all airlines redirects passengers to border control. Border control then determines who each passenger is according to passport credentials. Border control also checks whether the identified passenger is authorized to fly to the destination corresponding to the ticket, perhaps based on visa credentials. Then, at the departure gate, an agent enforces the authorization from border control, allowing the passenger to board the plane as long as the passenger has not gotten lost, or tried to board the wrong plane, or swapped tickets with someone else. Thus, border control handles access management at the airport.

OpenAM is most frequently used to protect web-accessible resources. Users browse to a protected web application page. An agent installed on the server with the web application redirects the user to OpenAM for access management. OpenAM determines who the user is, and whether the user has the right to access the protected page. OpenAM then redirects the user back to the protected page, with authorization credentials that can be verified by the agent. The agent allows OpenAM authorized users access the page.

Notice that OpenAM basically needs to determine two things for access management: the identity of the user, and whether the user has access rights to the protected page. *Authentication* is how OpenAM identifies the user. This chapter covers how to set up the authentication process. *Authorization* is how OpenAM determines whether a user has access to a protected resource. Authorization is covered later.

For authentication, OpenAM uses credentials from the user or client application. It then uses defined mechanisms to validate credentials and complete the authentication. The authentication methods can vary. For example, passengers travelling on international flights authenticate with passports and visas. In contrast, passengers travelling on domestic flights might authenticate with an identity card or a driver's license. Customers withdrawing cash from an ATM authenticate with a card and a PIN.

OpenAM allows you to configure authentication processes and then customize how they are applied. OpenAM uses *authentication modules* to handle different ways of authenticating. Basically, each

authentication module handles one way of obtaining and verifying credentials. You can chain different authentication modules together. In OpenAM, this is called *authentication chaining*. Each authentication module can be configured to specify the continuation and failure semantics with one of the following four flags: required, optional, requisite, or sufficient.<sup>1</sup>

- When a *required* module fails, the rest of the chain is processed, but the authentication fails.

A required module might be used for login with email and password, but then fall through to another module to handle new users who have not yet signed up.

- When an *optional* module fails, authentication continues.

An optional module might be used to permit a higher level of access if the user can present a X.509 certificate for example.

- When a *requisite* module fails, authentication fails and authentication processing stops.

A requisite module might be used with exclusive SSO.

- When a *sufficient* succeeds, authentication is successful and later modules in the chain are skipped.

You could set Windows Desktop SSO as sufficient, so authenticated Windows users are let through, whereas web users have to traverse another authentication module such as one requiring an email address and a password.

With OpenAM, you can further set *authentication levels* per module, with higher levels being used typically to allow access to more restricted resources. The OpenAM SPIs also let you develop your own authentication modules, and post-authentication plugins. Client applications can specify the authentication level, module, user, and authentication service to use among those you have configured. As described later in this guide, you can use *realms* to organize which authentication process applies for different applications or different domains, perhaps managed by different people.

When a user successfully authenticates, OpenAM creates a session, which allows OpenAM to manage that user's access to resources. In some deployments you need to limit how many active sessions a user can have at a given time. For example, you might want to prevent a user from using more than two devices at once. See Section 2.7, "Configuring Session Quotas" for instructions.

OpenAM leaves the authentication process flexible so that you can adapt how it works to your situation. Although at first the number of choices can seem daunting, now that you understand the basic process, you begin to see how choosing authentication modules and arranging them in authentication chains lets you use OpenAM to protect access to a wide range of applications used in your organization.

---

<sup>1</sup>The four flags, required, optional, requisite, and sufficient, come from the standards created for the Java Authentication and Authorization Service (JAAS).

## 2.2. Configuring Authentication Modules

The OpenAM console provides two places where the OpenAM administrator can configure authentication modules.

1. Under Configuration > Authentication, you configure available modules for use throughout OpenAM. What you set up here is inherited for use elsewhere.
2. Under Access Control > *Realm Name* > Authentication, you configure modules for your realm. What you set up at this level inherits from the global configuration, but you can override what is inherited. You can also add your own modules if necessary.

The configuration of individual modules depend on its function. The configuration of an Active Directory over LDAP user authentication module requires connection information and details about where to search for users. In contrast, the configuration of the HOTP module for OTP authentication requires data about the password length and the mail server or SMS gateway for to send the password during authentication.

### 2.2.1. Hints For the Active Directory Authentication Module

OpenAM connects to Active Directory over Lightweight Directory Access Protocol (LDAP). OpenAM provides separate Active Directory and LDAP modules to support the use of both Active Directory and another directory service in an authentication chain.

**ssoadm** service name: `sunAMAuthADService`

#### **Primary Active Directory Server** **Secondary Active Directory Server**

The default port for LDAP is 389. If you are connecting to Active Directory over SSL, the default port for LDAP/SSL is 636.

To allow users to change passwords through OpenAM, Active Directory requires that you connect over SSL.

If you want to use SSL or TLS for security, then scroll down to enable SSL/TLS Access to Active Directory Server. Make sure that OpenAM can trust the Active Directory certificate when using this option.

OpenAM first attempts to contact primary servers. If no primary server is available, then OpenAM attempts to contact secondaries.

When authenticating users from a directory server that is remote from OpenAM, set both the primary and secondary server values.

**ssoadm** attributes: primary is `iplanet-am-auth-ldap-server`; secondary is `iplanet-am-auth-ldap-server2`

## DN to Start User Search

LDAP data is organized hierarchically, a bit like a file system on Windows or UNIX. More specific DNs likely result in better performance. When configuring the module for a particular part of the organization, you can perhaps start searches from a specific organizational unit such as `OU=sales,DC=example,DC=com`.

If multiple entries exist with identical search attribute values, make this value specific enough to return only one entry.

**ssoadm** attribute: `iplanet-am-auth-ldap-base-dn`

## Bind User DN, Bind User Password

If OpenAM stores attributes in Active Directory, for example to manage account lockout, or if Active Directory requires that OpenAM authenticate in order to read users' attributes, then OpenAM needs the DN and password to authenticate to Active Directory.

The default is `amldapuser`. If the administrator authentication chain (default: `ldapService`) has been configured to include only the Active Directory module, then make sure that the password is correct before you logout. If it is incorrect, you will be locked out. If you do get locked out, you can login with the super user DN, which by default is `uid=amAdmin,ou=People,OpenAM-deploy-base`, where `OpenAM-deploy-base` was set during OpenAM configuration.

**ssoadm** attributes: `iplanet-am-auth-ldap-bind-dn` and `iplanet-am-auth-ldap-bind-passwd`

## Attributes Used to Retrieve User Profile

### Attributes Used to Search for a User to be Authenticated

#### User Search Filter

#### Search Scope

LDAP searches for user entries return entries with attribute values matching the filter you provide. For example if you search under `CN=Users,DC=example,DC=com` with a filter `"(MAIL=bjensen@example.com)"`, then the directory returns the entry that has `MAIL=bjensen@example.com`. In this example the attribute used to search for a user is `mail`. Multiple attribute values mean the user can authenticate with any one of the values. For example, if you have both `uid` and `mail`, then Barbara Jensen can authenticate with either `bjensen` or `bjensen@example.com`.

The User Search Filter text box provides a more complex filter. For example, if you search on `mail` and add User Search Filter `(objectClass=inetOrgPerson)`, then OpenAM uses the resulting search filter `(&(mail=address)(objectClass=inetOrgPerson))`, where `address` is the mail address provided by the user.

This controls how and the level of the directory that will be searched. You can set the search to run at a high level or against a specific area.

- OBJECT will search only for the entry specified as the DN to Start User Search.
- ONELEVEL will search only the entries that are directly children of that object.

- SUBTREE will search the entry specified and every entry under it.

**ssoadm** attributes: `iplanet-am-auth-ldap-user-naming-attribute`, `iplanet-am-auth-ldap-user-search-attributes`, `iplanet-am-auth-ldap-search-filter`, and `iplanet-am-auth-ldap-search-scope`

## SSL/TLS Access to Active Directory Server

If you enable SSL/TLS, OpenAM must be able to trust Active Directory certificates, either because the Active Directory certificates were signed by a CA whose certificate is already included in the trust store used by the container where OpenAM runs, or because you imported the certificates into the trust store.

**ssoadm** attribute: `iplanet-am-auth-ldap-ssl-enabled`

## Return User DN to DataStore

When enabled, and OpenAM uses Active Directory as the user store, the module returns the DN rather than the User ID, so the bind for authentication can be completed without a search to retrieve the DN.

**ssoadm** attribute: `iplanet-am-auth-ldap-return-user-dn`

## User Creation Attributes

This list lets you map (external) attribute names from Active Directory to (internal) attribute names used by OpenAM.

**ssoadm** attribute: `iplanet-am-ldap-user-creation-attr-list`

## LDAP Connection Heartbeat Interval

Specifies how often OpenAM should send a heartbeat request to the directory server to ensure that the connection does not remain idle. Some network administrators configure firewalls and load balancers to drop connections that are idle for too long. You can turn this off by setting the value to 0 or to a negative number. To set the units for the interval use LDAP Connection Heartbeat Time Unit.

Default: 1

**ssoadm** attribute: `openam-auth-ldap-heartbeat-interval`

## LDAP Connection Heartbeat Time Unit

Specifies the time unit corresponding to LDAP Connection Heartbeat Interval.

Default: minute

**ssoadm** attribute: `openam-auth-ldap-heartbeat-interval`

## Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `sunAMAuthADAuthLevel`

## LDAP Connection Operation Timeout

Defines the timeout in milliseconds that OpenAM should wait for a response from the directory server.

**ssoadm** attribute: `openam-auth-ldap-operation-timeout`

Default: 0 (means no timeout)

## 2.2.2. Hints For the Adaptive Risk Authentication Module

The Adaptive Risk module is designed to assess risk during authentication so that OpenAM can determine whether to require the user to complete further authentication steps. After configuring the Adaptive Risk module, insert it in your authentication chain with criteria set to sufficient as shown in the following example.

**AdaptiveRisk - Properties** Save Reset Back to Authentication

(3 Item(s))

Add Remove Reorder

<input checked="" type="checkbox"/>	Instance	Criteria	Options
<input type="checkbox"/>	LDAP	REQUIRED	
<input type="checkbox"/>	AdaptiveRisk	SUFFICIENT	
<input type="checkbox"/>	HOTP	REQUIRED	

In the example authentication chain shown, OpenAM has users authenticate first using the LDAP module providing a user ID and password combination. Upon success, OpenAM calls the Adaptive Risk module. The Adaptive Risk module assesses the risk based on your configured parameters. If the Adaptive Risk module calculates a total score below the threshold you set, the module returns success, and OpenAM finishes authentication processing without requiring further credentials. Otherwise the Adaptive Risk module evaluates the score to be above the risk threshold, and returns failure. OpenAM then calls the HOTP module, requiring the user to authenticate with a one-time password delivered to her by email or by SMS to her mobile phone.

When you configure the Adaptive Risk module to save cookies and profile attributes after successful authentication, OpenAM performs the save as post-authentication processing, only after the entire authentication chain returns success. You must set up OpenAM to save the data as part of post-authentication processing by editing the authentication chain to add `org.forgerock.openam.authentication.modules.adaptive.Adaptive` to the list of post authentication plugins.

**ssoadm** service name: `sunAMAuthAdaptiveService`

## General

### Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `openam-auth-adaptive-auth-level`

### Risk Threshold

Risk threshold score. If the sum of the Scores is greater than the threshold, the Adaptive Risk module returns failure. Default: 1

**ssoadm** attribute: `openam-auth-adaptive-auth-threshold`

## Failed Authentications

### Failed Authentication Check

When enabled, check the user profile for authentication failures since the last successful login. This check therefore requires OpenAM to have access to the user profile, and Account Lockout to be enabled (otherwise OpenAM does not record authentication failures).

**ssoadm** attribute: `openam-auth-adaptive-failure-check`

### Score

Value to add to the total score if the user fails the Failed Authentication Check. Default: 1

**ssoadm** attribute: `openam-auth-adaptive-failure-score`

### Invert Result

When selected, add the Score to the total score if the user passes the Failed Authentication Check.

**ssoadm** attribute: `openam-auth-adaptive-failure-invert`

## IP Address Range

### IP Range Check

When enabled, check whether the client IP address is within one of the specified IP Ranges.

**ssoadm** attribute: `openam-auth-adaptive-ip-range-check`

## IP Range

For IPv4, specifies a list of IP ranges either in CIDR-style notation ( $x.x.x.x/YY$ ) or as a range from one address to another ( $x.x.x.x-y.y.y.y$ , meaning from  $x.x.x.x$  to  $y.y.y.y$ ).

For IPv6, specifies a list of IP ranges either in CIDR-style notation ( $X:X:X:X:X:X/YY$ ) or as a range from one address to another ( $X:X:X:X:X:X-Y:Y:Y:Y:Y:Y$ , meaning from  $X:X:X:X:X:X$  to  $Y:Y:Y:Y:Y:Y$ ).

**ssoadm** attribute: `openam-auth-adaptive-ip-range-range`

## Score

Value to add to the total score if the user fails the IP Range Check. Default: 1

**ssoadm** attribute: `openam-auth-adaptive-ip-range-score`

## Invert Result

When selected, add the Score to the total score if the user passes the IP Range Check.

**ssoadm** attribute: `openam-auth-adaptive-ip-range-invert`

## IP Address History

### IP History Check

When enabled, check whether the client IP address matches one of the known values stored on the profile attribute you specify. This check therefore requires that OpenAM have access to the user profile.

**ssoadm** attribute: `openam-auth-adaptive-ip-history-check`

### History Size

Specifies how many IP address values to retain on the profile attribute you specify. Default: 5

**ssoadm** attribute: `openam-auth-ip-adaptive-history-count`

### Profile Attribute Name

Name of the user profile attribute on which to store known IP addresses. Default: `iphistory`

**ssoadm** attribute: `openam-auth-adaptive-ip-history-attribute`

### Save Successful IP Address

When enabled, save new client IP addresses to the known IP address list following successful authentication.

**ssoadm** attribute: `openam-auth-adaptive-ip-history-save`

### Score

Value to add to the total score if the user fails the IP History Check. Default: 1

**ssoadm** attribute: `openam-auth-adaptive-ip-history-score`

### Invert Result

When selected, add the Score to the total score if the user passes the IP History Check.

**ssoadm** attribute: `openam-auth-adaptive-ip-history-invert`

### Known Cookie

#### Cookie Value Check

When enabled, check whether the client browser request has the specified cookie and optional cookie value.

**ssoadm** attribute: `openam-auth-adaptive-known-cookie-check`

#### Cookie Name

Specifies the name of the cookie for which OpenAM checks when you enable the Cookie Value Check.

**ssoadm** attribute: `openam-auth-adaptive-known-cookie-name`

#### Cookie Value

Specifies the value of the cookie for which OpenAM checks. If no value is specified, OpenAM does not check the cookie value.

**ssoadm** attribute: `openam-auth-adaptive-known-cookie-value`

#### Save Cookie Value on Successful Login

When enabled, save the cookie as specified in the client's browser following successful authentication. If no Cookie Value is specified, the value is set to 1.

**ssoadm** attribute: `openam-auth-adaptive-known-cookie-save`

### Score

Value to add to the total score if user passes the Cookie Value Check. Default: 1

**ssoadm** attribute: `openam-auth-adaptive-known-cookie-score`

## Invert Result

When selected, add the Score to the total score if the user passes the Cookie Value Check.

**ssoadm** attribute: `openam-auth-adaptive-known-cookie-invert`

## Device Cookie

### Device Registration Cookie Check

When enabled, check whether the client browser request has the specified cookie with the correct device registration identifier as the value.

**ssoadm** attribute: `openam-auth-adaptive-device-cookie-check`

### Cookie Name

Specifies the name of the cookie for the Device Registration Cookie Check. Default: Device

**ssoadm** attribute: `openam-auth-adaptive-device-cookie-name`

### Save Device Registration on Successful Login

When enabled, save the specified cookie with a hashed device identifier value in the client's browser following successful authentication.

**ssoadm** attribute: `openam-auth-adaptive-device-cookie-save`

## Score

Value to add to the total score if the user fails the Device Registration Cookie Check. Default: 1

**ssoadm** attribute: `openam-auth-adaptive-device-cookie-score`

## Invert Result

When selected, add the Score to the total score if the user passes the Device Registration Cookie Check.

**ssoadm** attribute: `openam-auth-adaptive-device-cookie-invert`

## Time Since Last Login

### Time Since Last Login Check

When enabled, check whether the client browser request has the specified cookie that holds the encrypted last login time, and check that the last login time is more recent than a maximum number of days you specify.

**ssoadm** attribute: `openam-auth-adaptive-time-since-last-login-check`

## Cookie Name

Specifies the name of the cookie holding the encrypted last login time value.

**ssoadm** attribute: `openam-auth-adaptive-time-since-last-login-cookie-name`

## Max Time since Last Login

Specifies a threshold age of the last login time in days. If the client's last login time is more recent than the number of days specified, then the client successfully passes the check.

**ssoadm** attribute: `openam-auth-adaptive-time-since-last-login-value`

## Save time of Successful Login

When enabled, save the specified cookie with the current time encrypted as the last login value in the client's browser following successful authentication.

**ssoadm** attribute: `openam-auth-adaptive-time-since-last-login-save`

## Score

Value to add to the total score if the user fails the Time Since Last Login Check. Default: 1

**ssoadm** attribute: `openam-auth-adaptive-time-since-last-login-score`

## Invert Result

When selected, add the Score to the total score if the user passes the Time Since Last Login Check.

**ssoadm** attribute: `openam-auth-adaptive-time-since-last-login-invert`

## Profile Attribute

### Profile Risk Attribute check

When enabled, check whether the user profile contains the specified attribute and value.

**ssoadm** attribute: `openam-auth-adaptive-risk-attribute-check`

### Attribute Name

Specifies the attribute to check on the user profile for the specified value.

**ssoadm** attribute: `openam-auth-adaptive-risk-attribute-name`

### Attribute Value

Specifies the value to match on the profile attribute. If the attribute is multi-valued, a single match is sufficient to pass the check.

**ssoadm** attribute: `openam-auth-adaptive-risk-attribute-value`

### Score

Value to add to the total score if the user fails the Profile Risk Attribute Check. Default: 1

**ssoadm** attribute: `openam-auth-adaptive-risk-attribute-score`

### Invert Result

When selected, add the Score to the total score if the user passes the Profile Risk Attribute Check.

**ssoadm** attribute: `openam-auth-adaptive-risk-attribute-invert`

## Geo Location

### Geolocation Country Code Check

When enabled, check whether the client IP address location matches a country specified in the Valid Country Codes list. The

**ssoadm** attribute: `forgerock-am-auth-adaptive-geo-location-check`

### Geolocation Database location

Path to GeoIP data file used to convert IP addresses to country locations. The geolocation database is not packaged with OpenAM. You can download the GeoIP Country database from [MaxMind](#). Use the binary .dat file format, rather than .csv. You can use the GeoLite Country database for testing.

**ssoadm** attribute: `openam-auth-adaptive-geo-location-database`

### Valid Country Codes

Specifies the list of country codes to match. Use | to separate multiple values.

**ssoadm** attribute: `openam-auth-adaptive-geo-location-values.`

### Score

Value to add to the total score if the user fails the Geolocation Country Code Check. Default: 1

**ssoadm** attribute: `openam-auth-adaptive-geo-location-score`

### Invert Result

When selected, add the Score to the total score if the user passes the Geolocation Country Code Check.

**ssoadm** attribute: `openam-auth-adaptive-geo-location-invert`

## Request Header

### Request Header Check

When enabled, check whether the client browser request has the specified header with the correct value.

**ssoadm** attribute: `openam-auth-adaptive-req-header-check`

### Request Header Name

Specifies the name of the request header for the Request Header Check.

**ssoadm** attribute: `openam-auth-adaptive-req-header-name`

### Request Header Value

Specifies the value of the request header for the Request Header Check.

**ssoadm** attribute: `openam-auth-adaptive-req-header-value`

### Score

Value to add to the total score if the user fails the Request Header Check. Default: 1

**ssoadm** attribute: `openam-auth-adaptive-req-header-score`

### Invert Result

When selected, add the Score to the total score if the user passes the Request Header Check.

**ssoadm** attribute: `openam-auth-adaptive-req-header-invert`

## 2.2.3. Hints For the Anonymous Authentication Module

This module lets you track and manage anonymous users, perhaps forcing further authentication later when a user moves to access resources that require more protection.

**ssoadm** service name: `iPlanetAMAuthAnonymousService`

### Valid Anonymous Users

Specifies valid anonymous user IDs in addition to the default.

**ssoadm** attribute: `iplanet-am-auth-anonymous-users-list`

## Default Anonymous User Name

Specifies the user ID assigned by the module if the Valid Anonymous Users list is empty. Default: `anonymous`

**ssoadm** attribute: `iplanet-am-auth-anonymous-default-user-name`

## Case Sensitive User IDs

Determines whether case matters for anonymous user IDs.

**ssoadm** attribute: `iplanet-am-auth-anonymous-case-sensitive`

## Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `iplanet-am-auth-anonymous-auth-level`

## 2.2.4. Hints For the Certificate Authentication Module

X.509 digital certificates can enable secure authentication without the need for user names and passwords or other credentials. Certificate authentication can be handy to manage authentication by applications. If all certificates are signed by a recognized Certificate Authority (CA), then you might get away without additional configuration. If you need to look up public keys of OpenAM clients, this module can also look up public keys in an LDAP directory server.

When you store certificates and certificate revocation lists (CRL) in an LDAP directory service, you must configure both how to access the directory service and also how to look up the certificates and CRLs, based on the fields in the certificates that OpenAM clients present to authenticate.

Access to the LDAP server and how to search for users is similar to LDAP module configuration as in Section 2.2.12, "Hints For the LDAP Authentication Module". The primary difference is that, unlike for LDAP configuration, OpenAM retrieves the user identifier from a field in the certificate that the client application presents, then uses that identifier to search for the LDAP directory entry that holds the certificate, which should match the certificate presented. For example, if the Subject field of a typical certificate has a DN `C=FR, O=Example Corp, CN=Barbara Jensen`, and Barbara Jensen's entry in the directory has `cn=Barbara Jensen`, then you can use `CN=Barbara Jensen` from the Subject DN to search for the entry with `cn=Barbara Jensen` in the directory.

**ssoadm** service name: `iPlanetAMAuthCertService`

## Match Certificate in LDAP

When enabled, OpenAM searches for a match for the user's certificate in the LDAP directory. If a match is found and not revoked according to a CRL or OCSP validation, then authentication succeeds.

**ssoadm** attribute: `iplanet-am-auth-cert-check-cert-in-ldap`

## Subject DN Attribute Used to Search LDAP for Certificates

Indicates which attribute and value in the certificate Subject DN is used to find the LDAP entry holding the certificate.

Default: CN

**ssoadm** attribute: `iplanet-am-auth-cert-attr-check-ldap`

## Match Certificate to CRL

When enabled, OpenAM checks whether the certificate has been revoked according to a CRL in the LDAP directory.

**ssoadm** attribute: `iplanet-am-auth-cert-check-crl`

## Issuer DN Attribute Used to Search LDAP for CRLs

Indicates which attribute and value in the certificate Issuer DN is used to find the CRL in the LDAP directory.

Default: CN

If only one attribute is specified, the LDAP search filter used to find the CRL based on the Subject DN of the CA certificate is `(attr-name=attr-value-in-subject-DN)`.

For example, if the subject DN of the issuer certificate is `C=US, CN=Some CA, serialNumber=123456`, and the attribute specified is `CN`, then the LDAP search filter used to find the CRL is `(CN=Some CA)`.

In order to distinguish among different CRLs for the same CA issuer, specify multiple attributes separated by commas (,) in the same order they occur in the subject DN. When multiple attribute names are provided in a comma-separated list, the LDAP search filter used is `(cn=attr1=attr1-value-in-subject-DN,attr2=attr2-value-in-subject-DN,...,attrN=attrN-value-in-subject-DN)`.

For example, if the subject DN of the issuer certificate is `C=US, CN=Some CA, serialNumber=123456`, and the attributes specified are `CN,serialNumber`, then the LDAP search filter used to find the CRL is `(cn=CN=Some CA,serialNumber=123456)`.

**ssoadm** attribute: `iplanet-am-auth-cert-attr-check-crl`

## HTTP Parameters for CRL Update

Your certificate authority should provide the URL to use here, from which OpenAM can get CRL updates.

**ssoadm** attribute: `iplanet-am-auth-cert-param-get-crl`

## Match CA Certificate to CRL

When enabled, OpenAM checks the CRL against the CA certificate to ensure it has not been compromised.

**ssoadm** attribute: `sunAMValidateCACert`

## Cache CRLs in Memory

When enabled, CRLs will be cached.

**ssoadm** attribute: `openam-am-auth-cert-attr-cache-crl`

## Update CA CRLs from CRLDistributionPoint

When enabled, OpenAM updates CRLs from the LDAP directory store.

**ssoadm** attribute: `openam-am-auth-cert-update-crl`

## OCSP Validation

Enable this to use Online Certificate Status Protocol (OCSP) instead of CRLs to check certificates' revocation status.

If you enable this, you also must configure OSCP for OpenAM under Configuration > Server and Sites > Default Server Settings, or Configuration > Server and Sites > *Server Name* > Security.

**ssoadm** attribute: `iplanet-am-auth-cert-check-ocsp`

## LDAP Server Where Certificates are Stored

Identifies the LDAP server with certificates. Remember to specify URLs with appropriate port numbers (389 for unencrypted LDAP, 636 for LDAP over SSL). When configuring a secure connection, scroll down to enable Use SSL/TLS for LDAP Access.

**ssoadm** attribute: `iplanet-am-auth-cert-ldap-provider-url`

## LDAP Search Start or Base DN

Valid base DN for the LDAP search, such as `dc=example,dc=com`.

**ssoadm** attribute: `iplanet-am-auth-cert-start-search-loc`

## LDAP Server Authentication User, LDAP Server Authentication Password

If OpenAM stores attributes in the LDAP directory, for example to manage account lockout, or if the LDAP directory requires that OpenAM authenticate in order to read users' attributes, then OpenAM needs the DN and password to authenticate to the LDAP directory.

**ssoadm** attributes: `iplanet-am-auth-cert-principal-user`, and `iplanet-am-auth-cert-principal-passwd`

## Use SSL/TLS for LDAP Access

If you use SSL/TLS for LDAP access, OpenAM must be able to trust the LDAP server certificate.

**ssoadm** attribute: `iplanet-am-auth-cert-use-ssl`

## Certificate Field Used to Access User Profile

If the user profile is in a different entry from the user certificate, then this can be different from subject DN attribute used to find the entry with the certificate. When you select other, provide an attribute name in the Other Certificate Field Used to Access User Profile text box.

**ssoadm** attribute: `iplanet-am-auth-cert-user-profile-mapper`

## Other Certificate Field Used to Access User Profile

This field is only used if the Certificate Field Used to Access User Profile attribute is set to other. This field allows a custom certificate field to be used as the basis of the user search.

**ssoadm** attribute: `iplanet-am-auth-cert-user-profile-mapper-other`

## SubjectAltNameExt Value Type to Access User Profile

Use this if you want to look up the user profile from an RFC 822 style name, or a User Principal Name as used in Active Directory.

**ssoadm** attribute: `iplanet-am-auth-cert-user-profile-mapper-ext`

## Trusted Remote Hosts

Hosts trusted to send certificates to OpenAM, such as load balancers doing SSL termination, or OpenAM distributed authentication UI instances.

**ssoadm** attribute: `iplanet-am-auth-cert-gw-cert-auth-enabled`

## HTTP Header Name for Client Certificate

If you configure trusted hosts, specify the HTTP header name for the client certificate inserted by the trusted host.

**ssoadm** attribute: `sunAMHttpParamName`

## Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `iplanet-am-auth-cert-auth-level`

## 2.2.5. Hints For the Core Authentication Module

The Core module is a meta-module.

The Core module lets you set up the list of modules available, and specify what types of client applications can authenticate with which modules. It also lets you configure connection pools for access to directory servers, and whether to retain objects used during authentication for use during

logout. Furthermore, the Core module lets you set defaults used when configuring authentication in a particular realm.

The Core Authentication module is divided into seven sections:

- Section 2.2.5.1, "Core - Global Attributes"
- Section 2.2.5.2, "Core - Realm Attributes"
- Section 2.2.5.3, "Core - Persistent Cookie (Legacy)"
- Section 2.2.5.4, "Core - Account Lockout"
- Section 2.2.5.5, "Core - General"
- Section 2.2.5.6, "Core - Security"
- Section 2.2.5.7, "Core - Post Authentication Processing"

**ssoadm** service name: `iPlanetAMAuthService`

### 2.2.5.1. Core - Global Attributes

The Global Attributes includes the list of available modules, LDAP connection settings, authentication process options, and an option to make the XUI the default interface. The Global Attributes are defined in the *Authentication Configuration* in the *Reference* section of the OpenAM Reference Guide.

### 2.2.5.2. Core - Realm Attributes

This section of the Core Authentication module is on the Realm Attributes section page before options for Persistent Cookies.

#### **User Profile** **User Profile**

Whether a user profile needs to exist in the user data store, or should be created on successful authentication.

#### **Dynamic**

Specifies that on successful authentication the Authentication Service creates a user profile if one does not already exist. OpenAM then issues the SSOToken. OpenAM creates the user profile in the user data store configured for the realm.

#### **Dynamic with User Alias**

Specifies that on successful authentication the Authentication Service creates a user profile that contains the User Alias List attribute which defines one or more aliases that for mapping a user's multiple profiles.

## Ignored

Specifies that a user profile is not required for the Authentication Service to issue an SSOToken after a successful authentication.

## Required

Specifies that on successful authentication the user must have a user profile in the user data store configured for the realm in order for the Authentication Service to issue an SSOToken.

**ssoadm** attribute: `iplanet-am-auth-dynamic-profile-creation`

## User Profile Dynamic Creation Default Roles

Specifies the Distinguished Name (DN) of a role to be assigned to a new user whose profile is created when either of the Dynamic options is selected under the User Profile attribute. There are no default values. The role specified must be within the realm for which the authentication process is configured.

This role can be either an OpenAM or Sun DSEE role, but it cannot be a filtered role. If you wish to automatically assign specific services to the user, you have to configure the Required Services attribute in the User Profile.

**ssoadm** attribute: `iplanet-am-auth-default-role`

## Alias Search Attribute Name

After a user is successfully authenticated, the user's profile is retrieved. OpenAM first searches for the user based on the data store settings. If that fails to find the user, OpenAM will use the attributes listed here to lookup the user profile. This setting accepts any data store specific attribute name.

**ssoadm** attribute: `iplanet-am-auth-alias-attr-name`

### 2.2.5.3. Core - Persistent Cookie (Legacy)

This section of the Core Authentication module covers the Persistent Cookie options.

#### Note

Two methods are available in OpenAM to configure persistent cookies. The options described here and in Section 2.5, "Authenticating To OpenAM" specify one method. There is also a new module, described in Section 2.2.17, "Hints for the Persistent Cookie Module". If you want to set up persistent cookies, you are encouraged to use the new module. The options described here have no effect on that module.

## Persistent Cookie Mode

Determines whether users can return to their authenticated session after restarting the browser. When enabled, the persistent cookie can be used to reauthenticate until the persistent cookie

expires (as specified by the value of the Persistent Cookie Maximum Time attribute), or until the user explicitly logs out. By default, the Authentication Service uses only memory cookies (expires when the browser is closed).

The client must explicitly request a persistent cookie by adding `iPSPCookie=yes` as a parameter to the login URL. OpenAM sets a `DProPCookie` as described in Section 2.5, "Authenticating To OpenAM".

**ssoadm** attribute: `iplanet-am-auth-persistent-cookie-mode`

### Persistent Cookie Maximum Time

Specifies the interval after which a persistent cookie expires. The interval begins when the user's session is successfully authenticated. The maximum value is 2147483647 (in seconds, so a bit more than 68 years). The field accepts any integer value less than the maximum.

**ssoadm** attribute: `iplanet-am-auth-persistent-cookie-time`

### Persistent Cookie Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `openam-auth-pcookie-auth-level`

## 2.2.5.4. Core - Account Lockout

This section of the Core Authentication module includes details on how account lockouts can be configured.

### Login Failure Lockout Mode

Selecting this attribute enables a physical lockout. Physical lockout will inactivate an LDAP attribute (defined in the Lockout Attribute Name property) in the user's profile. This attribute works in conjunction with several other lockout and notification attributes.

**ssoadm** attribute: `iplanet-am-auth-login-failure-lockout-mode`

### Login Failure Lockout Count

Defines the number of attempts that a user has to authenticate, within the time interval defined in Login Failure Lockout Interval, before being locked out.

**ssoadm** attribute: `iplanet-am-auth-login-failure-count`

### Login Failure Lockout Interval

Defines the time in minutes during which failed login attempts are counted. If one failed login attempt is followed by a second failed attempt, within this defined lockout interval time, the

lockout count starts, and the user is locked out if the number of attempts reaches the number defined in Login Failure Lockout Count. If an attempt within the defined lockout interval time proves successful before the number of attempts reaches the number defined in Login Failure Lockout Count, the lockout count is reset.

**ssoadm** attribute: `iplanet-am-auth-login-failure-duration`

### Email Address to Send Lockout Notification

Specify one (or more) email address(es) to which notification is sent if a user lockout occurs.

Separate multiple addresses with spaces, and append `|locale|charset` to addresses for recipients in non-English locales.

**ssoadm** attribute: `iplanet-am-auth-lockout-email-address`

### Warn User After N Failures

The number of authentication failures after which OpenAM displays a warning message that the user will be locked out.

**ssoadm** attribute: `iplanet-am-auth-lockout-warn-user`

### Login Failure Lockout Duration

Defines how many minutes a user must wait after a lockout before attempting to authenticate again. Entering a value greater than 0 enables memory lockout and disables physical lockout. Memory lockout means the user's account is locked in memory for the number of minutes specified. The account is unlocked after the time period has passed.

**ssoadm** attribute: `iplanet-am-auth-lockout-duration`

### Lockout Duration Multiplier

Defines a value with which to multiply the value of the Login Failure Lockout Duration attribute for each successive lockout. For example, if Login Failure Lockout Duration is set to 3 minutes, and the Lockout Duration Multiplier is set to 2, the user is locked out of the account for 6 minutes. Once the 6 minutes has elapsed, if the user again provides the wrong credentials, the lockout duration is then 12 minutes. With the Lockout Duration Multiplier, the lockout duration is incrementally increased based on the number of times the user has been locked out.

**ssoadm** attribute: `sunLockoutDurationMultiplier`

### Lockout Attribute Name

Defines the LDAP attribute used for physical lockout. The default value is `inetuserstatus`, although the field in the OpenAM console is empty. The Lockout Attribute Value field must also contain an appropriate value.

**ssoadm** attribute: `iplanet-am-auth-lockout-attribute-name`

## Lockout Attribute Value

Specifies the action to take on the attribute defined in Lockout Attribute Name. The default value is `inactive`, although the field in the OpenAM console is empty. The Lockout Attribute Name field must also contain an appropriate value.

**ssoadm** attribute: `iplanet-am-auth-lockout-attribute-value`

## Invalid Attempts Data Attribute Name

Specifies the LDAP attribute used to hold the number of failed authentication attempts towards Login Failure Lockout Count.

**ssoadm** attribute: `sunAMAuthInvalidAttemptsDataAttrName`

## Store Invalid Attempts in Data Store

Enables the storage of information regarding failed authentication attempts as the value of the Invalid Attempts Data Attribute Name in the user data store. In order to store data in this attribute, the OpenAM schema has to be loaded. Information stored includes number of invalid attempts, time of last failed attempt, lockout time and lockout duration. Storing this information in the identity repository allows it to be shared among multiple instances of OpenAM.

**ssoadm** attribute: `sunStoreInvalidAttemptsInDS`

## 2.2.5.5. Core - General

This section of the Core Authentication module includes general options.

### Default Authentication Locale

Specifies the default language subtype to be used by the Authentication Service. The default value is `en_US`.

**ssoadm** attribute: `iplanet-am-auth-locale`

### Identity Types

Lists the type or types of identities used during a profile lookup. You can choose more than one to search on multiple types if you would like OpenAM to conduct a second lookup if the first lookup fails. Default: Agent and User

#### Agent

Searches for identities under your agents.

#### agentgroup

Searches for identities according to your established agent group.

### agentonly

Searches for identities only under your agents.

### Group

Searches for identities according to your established groups.

### User

Searches for identities according to your users.

**ssoadm** attribute: `sunAMIdentityType`

## Pluggable User Status Event Classes

Specifies one or more Java classes used to provide a callback mechanism for user status changes during the authentication process. The Java class must implement the `com.sun.identity.authentication.spi.AMAuthCallBack` OpenAM interface. OpenAM supports account lockout and password changes. OpenAM supports password changes through the LDAP authentication module, and so the feature is only available for the LDAP module.

A `.jar` containing the user status event class belongs in the `WEB-INF/lib` directory of the deployed OpenAM instance. If you do not build a `.jar`, add the class files under `WEB-INF/classes`.

**ssoadm** attribute: `sunAMUserStatusCallbackPlugins`

## Default Authentication Level

Specifies the default authentication level for authentication modules.

**ssoadm** attribute: `iplanet-am-auth-default-auth-level`

## 2.2.5.6. Core - Security

This section of the Core Authentication module includes basic security options.

### Module Based Authentication

Enables users to authenticate using module-based authentication. Otherwise, all attempts at authentication using the `module=module-name` login parameter result in failure. It is recommended that this be turned off in a production environment.

**ssoadm** attribute: `sunEnableModuleBasedAuth`

### Valid goto URL domains

List external domains to which clients can be redirected after authentication. This attribute requires valid DNS domains that reflect the set policy rules, such as `https://website.example.com/*` or `https://website.example.com/*?*`.

**ssoadm** attribute: `iplanet-am-auth-valid-goto-domains`

## Zero Page Login

Enables the administrator to allow users to authenticate to a single authentication screen using GET request parameters. Enable this with caution as it can allow credentials to be cached.

**ssoadm** attribute: `openam.auth.zero.page.login.enabled`

## 2.2.5.7. Core - Post Authentication Processing

This section of the Core Authentication module specifies options for post authentication processing.

### Default Success Login URL

Accepts a list of values that specifies where users are directed after successful authentication. The format of this attribute is `client-type|URL` although the only value you can specify at this time is a URL which assumes the type HTML. The default value is `/openam/console`. Values that do not specify HTTP have that appended to the deployment URI.

**ssoadm** attribute: `iplanet-am-auth-login-success-url`

### Default Failure Login URL

Accepts a list of values that specifies where users are directed after authentication has failed. The format of this attribute is `client-type|URL` although the only value you can specify at this time is a URL which assumes the type HTML. Values that do not specify HTTP have that appended to the deployment URI.

**ssoadm** attribute: `iplanet-am-auth-login-failure-url`

### Authentication Post Processing Classes

Specifies one or more Java classes used to customize post authentication processes for successful or unsuccessful logins. The Java class must implement the `com.sun.identity.authentication.spi.AMPostAuthProcessInterface` OpenAM interface.

A .jar containing the post processing class belongs in the `WEB-INF/lib` directory of the deployed OpenAM instance. If you do not build a .jar, add the class files under `WEB-INF/classes`. For deployment, add the .jar or classes into a custom OpenAM .war file.

**ssoadm** attribute: `iplanet-am-auth-post-login-process-class`

### Generate UserID Mode

When enabled, the Membership module generates a list of alternate user identifiers if the one entered by a user during the self-registration process is not valid or already exists. The user identifiers are generated by the class specified in the Pluggable User Name Generator Class property.

**ssoadm** attribute: `iplanet-am-auth-username-generator-enabled`

### Pluggable User Name Generator Class

Specifies the name of the class used to generate alternate user identifiers when Generate UserID Mode is enabled. The default value is `com.sun.identity.authentication.spi.DefaultUserIDGenerator`.

**ssoadm** attribute: `iplanet-am-auth-username-generator-class`

### User Attribute Mapping to Session Attribute

Enables the authenticating user's identity attributes (stored in the identity repository) to be set as session properties in the user's SSOToken. The value takes the format `User-Profile-Attribute|Session-Attribute-Name`. If `Session-Attribute-Name` is not specified, the value of `User-Profile-Attribute` is used. All session attributes contain the `am.protected` prefix to ensure that they cannot be edited by the Client SDK.

For example, if you define the user profile attribute as mail and the user's email address (available in the user session) as `user.mail`, the entry for this attribute would be `mail|user.mail`. After a successful authentication, the `SSOToken.getProperty(String)` method is used to retrieve the user profile attribute set in the session. The user's email address is retrieved from the user's session using the `SSOToken.getProperty("am.protected.user.mail")` method call.

Properties that are set in the user session using User Attribute Mapping to Session Attributes can not be modified (for example, `SSOToken.setProperty(String, String)`). This results in an `SSOException`. Multi-value attributes, such as `memberOf`, are listed as a single session variable with a `|` separator.

**ssoadm** attribute: `sunAMUserAttributesSessionMapping`

## 2.2.6. Hints For the Data Store Authentication Module

The Data Store authentication module allows a login using the Identity Repository of the realm to authenticate users. Using the Data Store module removes the requirement to write an authentication plug-in module, load, and then configure the authentication module if you need to authenticate against the same data store repository. Additionally, you do not need to write a custom authentication module where flat-file authentication is needed for the corresponding repository in that realm.

Yet, the Data Store module is generic. It does not implement data store-specific capabilities such as the password policy and password reset features provided by LDAP modules. Therefore the Data Store module returns failure when such capabilities are invoked.

**ssoadm** service name: `sunAMAAuthDataStoreService`

### Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `sunAMAAuthDataStoreAuthLevel`

## 2.2.7. Hints For the Device Print Authentication Module

The Device Print module is a device fingerprinter. It collects information about client device locations, fonts, plugins, and more through their browsers. This module does not stand on its own, as it uses authentication information from a service to validate a username. The Device Print module then validates other characteristics of the user's system.

If no device print profile is stored for a user, then the user is prompted to fill in an HMAC One-Time Password (HOTP), sent by email or text. The user can then save that device print profile. On subsequent requests the user's device profile is recalled. Each of the device print attributes from the client are compared against each of the stored attributes. If the differences exceed a configured number of penalty points, the user is asked for another HOTP.

If you know the HOTP module, some of the attributes in this section may seem familiar. That is intentional, as the Device Print module includes all HOTP attributes, though not in the same order.

### Note

The following example assumes that the Device Print module comes after LDAP authentication. Alternatively, you could set it up after another directory service module such as Active Directory, Data Store, RADIUS, or Windows NT.

#### DP\_Demo - Properties

Save Reset Back to Authentication

(2 Item(s))		
Instance	Criteria	Options
<input checked="" type="checkbox"/> LDAP	REQUIRED	
<input type="checkbox"/> DevicePrint	REQUIRED	

This table lists the authentication modules that make up this authentication chain.

A device fingerprint is based on the unique characteristics of a user like yourself and your associated device. You can configure the characteristics that should be part of the fingerprint. These characteristics can include:

- User agents, associated with the configuration of a web browser.
- Installed fonts.
- The plugins installed for the web browser.
- The resolution and color depth associated with a display
- The timezone or even the geolocation of a device.

You can specify **penalty points** when characteristics of the fingerprint have changed. If the total **penalty points** exceeds some configured score, the user may be asked to verify their identity using HOTP.

The default value for each `*-penalty-points` attribute is 35, with a `*-max-tolerated-penalty-points` of 50. In other words, if more than one characteristic of the device fingerprint has changed, the module assumes that the user is trying to connect from a different system.

**ssoadm** service name: `iPlanetAMAuthDevicePrintModuleService`

### Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `openam-auth-adaptive-auth-level`

### Profile expiration days

The list of device attributes is collected in a profile, which automatically expires in a given number of days. Default: 30.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-profile-expiration-days`

### Maximum stored profile quantity

The number of device profiles may be limited by user. Default: 5.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-maximum-profiles-stored-quantity`

### Automatically store new profiles

When enabled, new device profiles are automatically stored, once the HOTP is verified. Otherwise, the user is prompted for confirmation.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-store-profiles-without-confirmation`

### Default user profile name

Every device requires a default profile name that represents the user.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-default-profile-name`

### User agent penalty points

Specifies the number of penalty points added when there is a difference between the current and stored user agent. The module adds a fixed number of points if the web browser or related fields are changed. Version differences are ignored if the `User-Agent version ignore` attribute is active. Default: 35.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-user-agent-penalty-points`

### Installed fonts penalty points

Frequently, when a user installs a new software component, additional fonts are installed. The Device Print module adds a fixed number of penalty points if there is a change in the installed

fonts since the last login beyond some maximum tolerated level, and the `Installed fonts required` attribute is enabled. Default: 35.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-installed-fonts-penalty-points`

### Installed plugins penalty points

Most devices have a number of plugins installed on their browsers, such as Java, Flash, and document readers. If the plugins feature is enabled, and there is a change beyond a maximum tolerated level, a fixed number of penalty points is added. Default: 35.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-installed-plugins-penalty-points`

### Screen color depth penalty points

Most client devices are connected to a color monitor, or at least have settings related to such a device. If the `Screen parameters` feature is enabled, and a change in color depth is detected, a fixed number of penalty points is added. Default: 35.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-screen-color-depth-penalty-points`

### Screen resolution penalty points

Differences in screen resolution sometimes indicate that the user has changed monitors, or perhaps is adjusting to changing eyesight. Whenever a different screen resolution is detected, a fixed number of penalty points is added. Default: 35.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-screen-resolution-penalty-points`

### Timezone penalty points

Whenever a connection is made from a different timezone, the Device Print module normally adds a fixed number of penalty points. Organizations where most users travel may want to set this attribute to a lower value. Default: 35.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-timezone-penalty-points`

### Allowed location range

If geolocation is enabled, each profile should include a geographic location. The allowed location range specifies a maximum distance, in miles. If a user has travelled beyond that distance, the `Location penalty points` is added to the total. Default: 100.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-location-allowed-range`

### User-Agent version ignore

Upgrades change the version associated with a user agent. Unless this option is enabled, every change in the version of a browser or related fields is detected as a difference with the stored profile.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-ignore-version-in-user-agent`

### Max number of tolerated different installed fonts

If the `Installed fonts required` attribute is enabled, a comparison is made in installed fonts between the existing and stored profiles.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-max-tolerated-diffs-in-installed-fonts`

### Max tolerated percentage difference between installed fonts

If the `Installed fonts required` attribute is enabled, a comparison is made in installed fonts between the existing and stored profiles. Any differences in number of fonts is noted in percent.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-max-tolerated-percentage-to-mark-as-different-installed-fonts`

### Max number of tolerated different installed plugins

If the `Installed plugins required` attribute is enabled, a comparison is made between the installed plugins defined in the existing and stored profiles.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-max-tolerated-diffs-in-installed-plugins`

### Max tolerated percentage difference between installed plugins

If the `Installed plugins required` attribute is enabled, a comparison is made between the installed plugins defined in the existing and stored profiles. Any differences in number of plugins is noted in percent.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-max-tolerated-percentage-to-mark-as-different-plugins`

### User agent required

The user agent, as defined by RFC 4226 Section 14.43 provides information about the browser, and frequently on the operating system. Default: enabled.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-user-agent-required`

### Installed plugins required

Plugins are components that add a specific feature to an existing application. In this context, examples of browser plugins are flash, java, and shockwave.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-plugins-required`

### Installed fonts required

Fonts are frequently added to an operating system when new applications are installed. If enabled, installed font information is stored as part of the profile, and checked upon reconnection for comparison.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-fonts-required`

## Geolocation required

The geographic location of a device can be tracked. Can be used to limit access from this device to a specified range.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-geolocation-required`

## Screen parameters required

Display parameters can be used to help differentiate a profile. If enabled, the Device Print module uses screen color depth and resolution.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-screen-params-required`

## Time zone required

The time zone of the current and stored profiles can be collected and compared.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-timezone-required`

## Location penalty points

If geo-location has been enabled, and the current location does not match the value stored in the profile this many points are added as a penalty. Default: 35.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-location-penalty-points`

## Maximum tolerated penalty points

A limit, in number of points, between the current and stored Device Print profiles. If the total is below that value, the Device Print module sees a match. Default: 50.

**ssoadm** attribute: `iplanet-am-auth-adaptive-device-print-max-tolerated-penalty-points`

## SMS Gateway Implementation Class

Change this if you must customize the SMS gateway implementation. The default class sends an SMS or email, depending on the configuration.

**ssoadm** attribute: `sunAMAuthHOTPSMSGatewayImplClassName`

## Mail Server Host Name

Host name of the mail server supporting Simple Message Transfer Protocol for electronic mail.

**ssoadm** attribute: `sunAMAuthHOTPSMTPHostName`

## Mail Server Host Port

The default outgoing mail server port is 25, 465 (when connecting over SSL).

**ssoadm** attribute: `sunAMAuthHOTPSMTPHostPort`

## Mail Server Authentication Username

User name for OpenAM to connect to the mail server.

**ssoadm** attribute: `sunAMAuthHOTPSMTPUserName`

## Mail Server Authentication Password

Password for OpenAM to connect to the mail server.

**ssoadm** attribute: `sunAMAuthHOTPSMTPUserPassword`

## Mail Server Secure Connection

If OpenAM connects to the mail server securely, OpenAM must be able to trust the server certificate.

**ssoadm** attribute: `sunAMAuthHOTPSMTPSSLEnabled`

## Email From Address

The **From:** address when sending a one-time password by mail.

**ssoadm** attribute: `sunAMAuthHOTPSMTPFromAddress`

## One Time Password Validity Length (in minutes)

One-time passwords are valid for 5 minutes after they are generated by default.

**ssoadm** attribute: `sunAMAuthHOTPPasswordValidityDuration`

## One Time Password Length (in digits)

Set the length of the one-time password to 6 or 8 digits.

**ssoadm** attribute: `sunAMAuthHOTPPasswordLength`

## One Time Password Delivery

Send the one-time password by SMS, by mail, or both.

**ssoadm** attribute: `sunAMAuthHOTPPasswordDelivery`

## Mobile Phone Number Attribute Names

Provides the attribute name used for the text message. The default value is `telephoneNumber`.

**ssoadm** attribute: `openamTelephoneAttribute`

## Mobile Carrier Attribute Name

Provides the name of the carrier that will send the text message.

Every carrier has their own attribute name ending, for example Verizon uses @vtext.com or vtext.com. Contact your mobile carrier to find out what their attribute name is. If you will be sending international texts, ask your carrier if a country code is a required.

**ssoadm** attribute: `openamSMSCarrierAttribute`

### Email Attribute Name

Provides the attribute name used to email the OTP. The default value is `mail` (email).

**ssoadm** attribute: `openamEmailAttribute`

### Auto Send OTP Code

Setup the HOTP module to automatically generate an email or text message when users begin the login process.

**ssoadm** attribute: `sunAMAuthHOTPAutoClicking`

## 2.2.8. Hints For the Federation Authentication Module

The Federation authentication module is used by a service provider to create a user session after validating single sign-on protocol messages. This authentication module is used by the SAML, SAMLv2, ID-FF, and WS-Federation protocols.

**ssoadm** service name: `sunAMAuthFederationService`

### Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `sunAMAuthFederationAuthLevel`

## 2.2.9. Hints For the HOTP Authentication Module

The HMAC One-Time Password authentication module works together with the Data Store module to retrieve a user's mail address or telephone number to send a one-time password to complete authentication.

To use HOTP you set up an authentication chain with the Data Store module as the `requisite` first module, and the HOTP module as the second `requisite` module. When authentication succeeds against the Data Store module, OpenAM passes the Email Address and Telephone Number attributes from the user profile to the HOTP module. For the HOTP module to use either attribute, the Email Address must contain a valid email address, or the Telephone Number must contain a valid SMS telephone number.

You can set the HOTP module to automatically generate a password when users begin logging into the system. You can also setup a mobile phone, mobile carrier, and email attributes for tighter

controls over where the messages are generated and what provider the messages go through to reach the user.

**ssoadm** service name: `sunAMAuthHOTPService`

### Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `sunAMAuthHOTPAuthLevel`

### SMS Gateway Implementation Class

Change this if you must customize the SMS gateway implementation. The default class sends an SMS or email, depending on the configuration.

**ssoadm** attribute: `sunAMAuthHOTPSSMSGatewayImplClassName`

### Mail Server Host Name

Host name of the mail server supporting Simple Message Transfer Protocol for electronic mail.

**ssoadm** attribute: `sunAMAuthHOTPSMTPHostName`

### Mail Server Host Port

The default outgoing mail server port is 25, 465 (when connecting over SSL).

**ssoadm** attribute: `sunAMAuthHOTPSMTPHostPort`

### Mail Server Authentication Username

User name for OpenAM to connect to the mail server.

**ssoadm** attribute: `sunAMAuthHOTPSMTPUserName`

### Mail Server Authentication Password

Password for OpenAM to connect to the mail server.

**ssoadm** attribute: `sunAMAuthHOTPSMTPUserPassword`

### Mail Server Secure Connection

If OpenAM connects to the mail server securely, OpenAM must be able to trust the server certificate.

**ssoadm** attribute: `sunAMAuthHOTPSMTPSSLEnabled`

## Email From Address

The **From:** address when sending a one-time password by mail.

**ssoadm** attribute: `sunAMAuthHOTPSMTPFromAddress`

## One Time Password Validity Length (in minutes)

One-time passwords are valid for 5 minutes after they are generated by default.

**ssoadm** attribute: `sunAMAuthHOTPPasswordValidityDuration`

## One Time Password Length (in digits)

Set the length of the one-time password to 6 or 8 digits.

**ssoadm** attribute: `sunAMAuthHOTPPasswordLength`

## One Time Password Delivery

Send the one-time password by SMS, by mail, or both.

**ssoadm** attribute: `sunAMAuthHOTPPasswordDelivery`

## Mobile Phone Number Attribute Names

Provides the attribute name used for the text message. The default value is `telephoneNumber`.

**ssoadm** attribute: `openamTelephoneAttribute`

## Mobile Carrier Attribute Name

Provides the name of the carrier that will send the text message.

Every carrier has their own attribute name ending, for example Verizon uses `@vtext.com` or `vtext.com`. Contact your mobile carrier to find out what their attribute name is. If you will be sending international texts, ask your carrier if a country code is a required.

**ssoadm** attribute: `openamSMSCarrierAttribute`

## Email Attribute Name

Provides the attribute name used to email the OTP. The default value is `mail` (email).

**ssoadm** attribute: `openamEmailAttribute`

## Auto Send OTP Code

Setup the HOTP module to automatically generate an email or text message when users begin the login process.

**ssoadm** attribute: `sunAMAuthHOTPAutoClicking`

## 2.2.10. Hints For the HTTP Basic Authentication Module

HTTP basic authentication takes a user name and password from HTTP authentication and tries authentication against the backend module in OpenAM, depending on what you configure as the Backend Module Name.

**ssoadm** service name: `iPlanetAMAuthHTTPBasicService`

### Backend Module Name

Specifies the module that checks the user credentials. The credentials are then supplied to either a Data Store or other identity repository module for authentication.

**ssoadm** attribute: `iplanet-am-auth-http-basic-module-configured`

### Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `iplanet-am-auth-httpbasic-auth-level`

## 2.2.11. Hints For the JDBC Authentication Module

The Java Database Connectivity (JDBC) module lets OpenAM connect to a database such as MySQL or Oracle DB to authenticate users.

**ssoadm** service name: `sunAMAuthJDBCService`

### Connection Type

Choose Connection pool is retrieved via JNDI to connect using the Java Naming and Directory Interface connection pool supported by the web container in which OpenAM runs. Choose Non-persistent JDBC connection to connect directly through the JDBC driver.

**ssoadm** attribute: `sunAMAuthJDBCConnectionType`

### Connection Pool JNDI Name

When using Connection pool is retrieved via JNDI, this specifies the pool. How you configure connection pooling depends on the web container where you run OpenAM. Refer to the documentation for your web container for instructions on setting up connection pooling.

**ssoadm** attribute: `sunAMAuthJDBCJndiName`

### JDBC Driver

When using Non-persistent JDBC connection, this specifies the JDBC driver provided by the database.

The .jar containing the JDBC driver belongs in the `WEB-INF/lib` directory of the deployed OpenAM instance, and so you should add it to a custom OpenAM .war file that you deploy.

**ssoadm** attribute: `sunAMAuthJDBCdriver`

## JDBC URL

When using Non-persistent JDBC connection, this specifies the URL to connect to the database.

**ssoadm** attribute: `sunAMAuthJDBCurl`

## Database Username

Specify the user name to open the database connection.

**ssoadm** attribute: `sunAMAuthJDBCdbuser`

## Database Password

Specify the password for the user opening the database connection.

**ssoadm** attribute: `sunAMAuthJDBCdbpassword`

## Password Column Name

Specify the database column name where passwords are stored.

**ssoadm** attribute: `sunAMAuthJDBCPasswordColumn`

## Prepared Statement

Specify the SQL query to return the password corresponding to the user to authenticate.

**ssoadm** attribute: `sunAMAuthJDBCStatement`

## Class to Transform Password Syntax

Specify the class that transforms the password retrieved to the same format as provided by the user.

The default class expects the password in clear text. Custom classes must implement the `JDBCPasswordSyntaxTransform` interface.

**ssoadm** attribute: `sunAMAuthJDBCPasswordSyntaxTransformPlugin`

## Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `sunAMAuthJDBCAuthLevel`

## 2.2.12. Hints For the LDAP Authentication Module

OpenAM connects to directory servers using Lightweight Directory Access Protocol (LDAP). To build an easy-to-manage, high performance, pure Java, open source directory service, try OpenDJ directory services.

**ssoadm** service name: `iPlanetAMAuthLDAPService`

### Primary LDAP Server Secondary LDAP Server

Directory servers generally use built-in data replication for high availability. Thus a directory service likely consists of a pool of replicas to which OpenAM can connect to retrieve and update directory data. You set up primary and secondary servers in case a replica is down due to maintenance or to a problem with a particular server.

Set one primary and optionally one secondary directory server for each OpenAM server. For the current OpenAM server, specify each directory server as a `host:port` combination. For other OpenAM servers in the deployment, you can specify each directory server as `server-name|host:port`, where `server-name` is the FQDN portion of the OpenAM server from the list under Configuration > Servers and Sites, and `host:port` identifies the directory server.

For example, if the `server-name` that is listed is `http://openam.example.com:8080/openam` and the directory server is accessible at `opendj.example.com:1389`, you would enter `openam.example.com|opendj.example.com:1389`.

When authenticating users from a directory service that is remote from OpenAM, set both the primary and secondary server values.

If you want to use SSL or TLS for security, then scroll down to enable SSL/TLS Access to LDAP Server. Make sure that OpenAM can trust the servers' certificates when using this option.

**ssoadm** attributes: primary is `iplanet-am-auth-ldap-server`, secondary is `iplanet-am-auth-ldap-server2`, and `iplanet-am-auth-ldap-ssl-enabled`

### DN to Start User Search

LDAP data is organized hierarchically, a bit like a file system on Windows or UNIX. More specific DNs likely result in better search performance. When configuring the module for a particular part of the organization, you can perhaps start searches from a specific organizational unit such as `ou=sales,dc=example,dc=com`.

If multiple entries exist with identical search attribute values, make this value specific enough to return only one entry.

**ssoadm** attribute: `iplanet-am-auth-ldap-base-dn`

## Bind User DN, Bind User Password

If OpenAM stores attributes in the directory, for example to manage account lockout, or if the directory requires that OpenAM authenticate in order to read users' attributes, then OpenAM needs the DN and password to authenticate to the directory.

The default is `cn=Directory Manager`. Make sure that password is correct before you logout. If it is incorrect, you will be locked out. If this should occur, you can login with the super user DN, which by default is `uid=amAdmin,ou=People,OpenAM-deploy-base`, where *OpenAM-deploy-base* you set during OpenAM configuration.

**ssoadm** attributes: `iplanet-am-auth-ldap-bind-dn, iplanet-am-auth-ldap-bind-passwd`

## Attributes Used to Retrieve User Profile

### Attributes Used to Search for a User to be Authenticated

#### User Search Filter

#### Search Scope

LDAP searches for user entries return entries with attribute values matching the filter you provide. For example if you search under `ou=people,dc=example,dc=com` with a filter `"(mail=bjensen@example.com)"`, then the directory returns the entry that has `mail=bjensen@example.com`. In this example the attribute used to search for a user is `mail`. Multiple attribute values mean the user can authenticate with any one of the values. For example, if you have both `uid` and `mail`, then Barbara Jensen can authenticate with either `bjensen` or `bjensen@example.com`.

Should you require a more complex filter for performance, you add that to the User Search Filter text box. For example, if you search on `mail` and add User Search Filter `(objectClass=inetOrgPerson)`, then OpenAM uses the resulting search filter `(&(mail=address)(objectClass=inetOrgPerson))`, where *address* is the mail address provided by the user.

Scope OBJECT means search only the entry specified as the DN to Start User Search, whereas ONELEVEL means search only the entries that are directly children of that object. SUBTREE means search the entry specified and every entry under it.

**ssoadm** attributes: `iplanet-am-auth-ldap-user-naming-attribute, iplanet-am-auth-ldap-user-search-attributes, iplanet-am-auth-ldap-search-filter, and iplanet-am-auth-ldap-search-scope`

## SSL/TLS Access to LDAP Server

If you enable SSL/TLS, OpenAM must be able to trust LDAP certificates, either because the certificates were signed by a CA whose certificate is already included in the trust store used by the container where OpenAM runs, or because you imported the certificates into the trust store.

**ssoadm** attribute: `iplanet-am-auth-ldap-ssl-enabled`

## Return User DN to Authenticate

When enabled, and OpenAM uses the directory service as the user store, the module returns the DN rather than the rather than the User ID, so the bind for authentication can be completed without a search to retrieve the DN.

**ssoadm** attribute: `iplanet-am-auth-ldap-return-user-dn`

### User Creation Attributes

This list lets you map (external) attribute names from Active Directory to (internal) attribute names used by OpenAM.

**ssoadm** attribute: `iplanet-am-ldap-user-creation-attr-list`

### Minimum Password Length

Specify the minimum acceptable password length.

**ssoadm** attribute: `iplanet-am-auth-ldap-min-password-length`

### LDAP Behera Password Policy Support

When enabled, support interoperability with servers that implement the Internet-Draft, Password Policy for LDAP Directories.

Support for this Internet-Draft is limited to the LDAP authentication module. Other components of OpenAM, such as the password change functionality in the `/idm/EndUser` page, do not support the Internet-Draft. In general, outside of the LDAP authentication module, OpenAM binds to the directory server as an administrator, such as Directory Manager. When OpenAM binds to the directory server as an administrator rather than as an end user, many features of the Internet-Draft password policies do not apply.

**ssoadm** attribute: `iplanet-am-auth-ldap-behera-password-policy-enabled`

### Trust All Server Certificates

When enabled, blindly trust server certificates, including self-signed test certificates.

**ssoadm** attribute: `iplanet-am-auth-ldap-ssl-trust-all`

### LDAP Connection Heartbeat Interval

Specifies how often OpenAM should send a heartbeat request to the directory server to ensure that the connection does not remain idle. Some network administrators configure firewalls and load balancers to drop connections that are idle for too long. You can turn this off by setting the value to 0 or to a negative number. To set the units for the interval use LDAP Connection Heartbeat Time Unit.

Default: 1

**ssoadm** attribute: `openam-auth-ldap-heartbeat-interval`

### LDAP Connection Heartbeat Time Unit

Specifies the time unit corresponding to LDAP Connection Heartbeat Interval.

Default: minute

**ssoadm** attribute: `openam-auth-ldap-heartbeat-interval`

### Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `iplanet-am-auth-ldap-auth-level`

### LDAP Connection Operation Timeout

Defines the timeout in milliseconds that OpenAM should wait for a response from the directory server.

**ssoadm** attribute: `openam-auth-ldap-operation-timeout`

Default: 0 (means no timeout)

## 2.2.13. Hints For the Membership Authentication Module

The Membership module permits self-registration for new users. You can then have OpenAM create new user profiles in the identity repository.

**ssoadm** service name: `iPlanetAMAuthMembershipService`

### Minimum Password Length

Specify the minimum acceptable number of characters in the password provided during self-registration.

**ssoadm** attribute: `iplanet-am-auth-membership-min-password-length`

### Default User Roles

Specifies the Distinguished Name (DN) of a role to be assigned to a new user whose profile is created. There are no default values. The role specified must be within the realm for which the authentication process is configured.

This role can be either an OpenAM or Sun DSEE role, but it cannot be a filtered role. If you wish to automatically assign specific services to the user, you have to configure the Required Services attribute in the User Profile.

**ssoadm** attribute: `iplanet-am-auth-membership-default-roles`

### User Status After Registration

If you choose Inactive, then the new user has no access to services until an administrator activates the account.

**ssoadm** attribute: `iplanet-am-auth-membership-default-user-status`

## Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `iplanet-am-auth-membership-auth-level`

## 2.2.14. Hints For the MSISDN Authentication Module

The Mobile Station Integrated Services Digital Network (MSISDN) authentication module enables non-interactive authentication using a mobile subscriber ISDN associated with a terminal such as a mobile phone. The module checks the subscriber ISDN against the value found on a user's entry in an LDAP directory service.

**ssoadm** service name: `sunAMAuthMSISDNService`

### Trusted Gateway IP Address

Specifies a list of IP addresses of trusted clients that can access MSISDN modules. Either restrict the clients allowed to access the MSISDN module by adding each IPv4 or IPv6 address here, or leave the list empty to allow all clients to access the module. If you specify the value `none`, no clients are allowed access.

**ssoadm** attribute: `sunAMAuthMSISDNTrustedGatewayList`

### MSISDN Number Search Parameter Name

Specifies a list of parameter names that identify which parameters to search in the request header or cookie header for the MSISDN number. For example, if you define `x-Cookie-Param`, `AM_NUMBER`, and `COOKIE-ID`, the MSISDN authentication service checks those parameters for the MSISDN number.

**ssoadm** attribute: `sunAMAuthMSISDNParameterNameList`

### LDAP Server and Port

If you want to use SSL or TLS for security, then scroll down to enable SSL/TLS Access to LDAP. Make sure that OpenAM can trust the servers' certificates when using this option.

**ssoadm** attribute: `sunAMAuthMSISDNLdapProviderUrl`

### LDAP Start Search DN

Specify the DN of the entry where the search for the user's MSISDN number should start.

**ssoadm** attribute: `sunAMAuthMSISDNBaseDn`

## Attribute To Use To Search LDAP

Specify the name of the attribute in the user's profile that contains the MSISDN number to search for the user. The default is `sunIdentityMSISDNNumber`.

**ssoadm** attribute: `sunAMAuthMSISDNUserSearchAttribute`

## LDAP Server Authentication User

If OpenAM must authenticate to the directory server in order to search, then specify the bind DN. The default is `cn=amldapuser,ou=DSAME Users,dc=example,dc=com`.

**ssoadm** attribute: `sunAMAuthMSISDNPrincipalUser`

## LDAP Server Authentication Password

Specify the password corresponding to the bind DN.

**ssoadm** attribute: `sunAMAuthMSISDNPrincipalPasswd`

## SSL/TLS for LDAP Access

If you choose to enable SSL or TLS, then make sure that OpenAM can trust the servers' certificates.

**ssoadm** attribute: `sunAMAuthMSISDNUseSsl`

## MSISDN Header Search Attribute

Specify the headers to use for searching the request for the MSISDN number.

- Cookie Header tells OpenAM to search the cookie.
- Request Header tells OpenAM to search the request header.
- Request Parameter tells OpenAM to search the request parameters.

**ssoadm** attribute: `sunAMAuthMSISDNHeaderSearch`

## LDAP Attribute Used to Retrieve User Profile

Specify the LDAP attribute that is used during a search to return the user profile for MSISDN authentication service. The default is `uid`.

**ssoadm** attribute: `sunAMAuthMSISDNUserNamingAttribute`

## Return User DN to DataStore

Enable this option only when the OpenAM directory is the same as the directory configured for MSISDN searches. When enabled, this option allows the authentication module to return the DN

instead of the User ID. OpenAM thus does not need to perform an additional search with the user ID to find the user's entry.

**ssoadm** attribute: `sunAMAuthMSISDNReturnUserDN`

## Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `sunAMAuthMSISDNAuthLevel`

## 2.2.15. Hints For the OATH Module

The Open Authentication (OATH) module provides a more secure method for users to access their accounts with the help of a device, such as their mobile phone or Yubikey. Users can log into OpenAM and update their information more securely from a one-time password (OTP) displayed on their device. The OATH module includes the OATH standard protocols (RFC 4226 and RFC 6238). The OATH module has several enhancements to the HMAC One-Time Password (HOTP) Authentication Module, but does not replace the original module for those already using HOTP prior to the 10.1.0 release. The OATH module includes HOTP authentication and Time-Based One-Time Password (TOTP) authentication. Both types of authentication require an OATH compliant device that can provide the OTP.

HOTP authentication generates the OTP every time the user requests a new OTP on their device. The device tracks the number of times the user requests a new OTP, called the counter. The OTP displays for a period of time you designate in the setup, so the user may be further in the counter on their device than on their account. OpenAM will resynchronize the counter when the user finally logs in. To accommodate this, you set the number of passwords a user can generate before their device cannot be resynchronized. For example, if you set the number of HOTP Window Size to 50 and someone presses the button 30 on the user's device to generate a new OTP, the counter in OpenAM will review the OTPs until it reaches the OTP entered by the user. If someone presses the button 51 times, you will need to reset the counter to match the number on the device's counter before the user can login to OpenAM. HOTP authentication does not check earlier passwords, so if the user attempts to reset the counter on their device, they will not be able to login until you reset the counter in OpenAM to match their device.

TOTP authentication constantly generates a new OTP based on a time interval you specify. The device tracks the last two passwords generated and the current password. The Last Login Time monitors the time when a user logs in to make sure that user is not logged in several times within the present time period. Once a user log into OpenAM, they must wait for the time it takes TOTP to generate the next two passwords and display them. This prevents others from being able to access the users account using the OTP they entered. The user's account can be accessed again after the generation of the third new OTP is generated and displayed on their device. For this reason, the TOTP Time-Step Interval should not be so long as to lock users out, with a recommended time of 30 seconds.

An authentication chain can be created to generate an OTP from either HOTP or TOTP.

**ssoadm** service name: `iPlanetAMAuthOATHService`

### Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `iplanet-am-auth-oath-auth-level`

### One Time Password Length (in digits)

Set the length of the OTP between 6 and 9 digits long. The default value is 6 digits.

**ssoadm** attribute: `iplanet-am-auth-oath-password-length`

### Minimum Secret Key Length

The minimal number of characters required to set the Secret Key.

**ssoadm** attribute: `iplanet-am-auth-oath-min-secret-key-length`

### Secret Key Attribute Name

The name of the attribute where the key will be stored in the user profile.

**ssoadm** attribute: `iplanet-am-auth-oath-secret-key-attribute`

### OATH Algorithm to Use

Select whether to use HOTP or TOTP. You can create an authentication chain to allow for a greater variety of devices. The default value is HOTP.

**ssoadm** attribute: `iplanet-am-auth-oath-algorithm`

### HOTP Window Size

The number of requests that the system and the device can be off to resynchronize the password. If a user passes this number of requests before logging into the system, the password will not work. The default value is 100.

**ssoadm** attribute: `iplanet-am-auth-oath-hotp-window-size`

### Counter Attribute Name

The name of the HOTP attribute where the counter will be stored in the user profile.

**ssoadm** attribute: `iplanet-am-auth-oath-hotp-counter-attribute`

### Add Checksum Digit

Adds a checksum digit at the end of the HOTP password to verify it was entered correctly. The default value is No.

**ssoadm** attribute: `iplanet-am-auth-oath-add-checksum`

### Truncation Offset

Advanced feature that is device specific. Any value below 0 or above 15 will turn off the functionality. The default value is -1. If not required by the device, leave at the default setting.

**ssoadm** attribute: `iplanet-am-auth-oath-truncation-offset`

### TOTP Time Step Interval

Defines how long the password will appear on the user's device (in seconds). We recommend keeping this number low, for example 30 seconds, because once a user logs out, they will not be able to login again until two full time cycles have passed. The default value is 30 seconds.

**ssoadm** attribute: `iplanet-am-auth-oath-size-of-time-step`

### TOTP Time Steps

The number of requests that the system and the device can be off to resynchronize the password. If a user passes this number of requests before logging into the system, the password will not work. The default value is 2.

**ssoadm** attribute: `iplanet-am-auth-oath-steps-in-window`

### Last Login Time Attribute

The name of the attribute where both HOTP and TOTP authentication will store information on when a person last logged in.

**ssoadm** attribute: `iplanet-am-auth-oath-last-login-time-attribute-name`

If you plan to use Yubikey for your OATH module, you will need to take some time to set it up. Go to the [YubiKey website](#) to configure your YubiKey device. If you do not have a YubiKey device, you can purchase them from this page as well.

Select the appropriate instructions, keeping in mind that the cross-platform personalization tool is recommended unless you have specific need for one of the other types. You have the greatest selection of platforms for this choice. Each device will need to be setup before use.

## 2.2.16. Hints For the OAuth 2.0 Authentication Module

The OAuth 2.0 authentication module lets OpenAM authenticate clients of OAuth resource servers. References in this section are to RFC 6749, The OAuth 2.0 Authorization Framework.

#### Note

The default settings are for Facebook.

**ssoadm** service name: `sunAMAuthOAuthService`

## Client ID

OAuth `client_id` as described in section 2.2 of RFC 6749.

**ssoadm** attribute: `iplanet-am-auth-oauth-client-id`

## Client Secret

OAuth `client_secret` as described in section 2.3 of RFC 6749.

**ssoadm** attribute: `iplanet-am-auth-oauth-client-secret`

## Authentication Endpoint URL

URL to the end point handling OAuth authentication as described in section 3.1 of RFC 6749. The default value is `https://www.facebook.com/dialog/oauth`.

**ssoadm** attribute: `iplanet-am-auth-oauth-auth-service`

## Access Token Endpoint URL

URL to the end point handling access tokens as described in section 3.2 of RFC 6749. The default value is `https://graph.facebook.com/oauth/access_token`.

**ssoadm** attribute: `iplanet-am-auth-oauth-token-service`

## User Profile Service URL

User profile URL that returns profile information in JSON format. The default value is `https://graph.facebook.com/me`.

**ssoadm** attribute: `iplanet-am-auth-oauth-user-profile-service`

## Scope

Comma separated list of user profile attributes that the application requires. The default value is `email,read_stream`.

**ssoadm** attribute: `iplanet-am-auth-oauth-scope`

## Proxy URL

URL to the `/oauth2c/0AuthProxy.jsp` file, part of OpenAM.

**ssoadm** attribute: `iplanet-am-auth-oauth-ss-proxy-url`

## Account Mapper

Class implementing account mapping. The default value is `org.forgerock.openam.authentication.modules.oauth2.DefaultAccountMapper`.

**ssoadm** attribute: `org-forgerock-auth-oauth-account-mapper`

## Account Mapper Configuration

Map of OAuth Provider user account attributes used to find the local profile of the authenticated user, with values in the form `provider-attr=local-attr`. Default values `email=mail` and `id=facebook-id`.

**ssoadm** attribute: `org-forgerock-auth-oauth-account-mapper-configuration`

## Attribute Mapper

Class implementing attribute mapping. Default: `org.forgerock.openam.authentication.modules.oauth2.DefaultAttributeMapper`

**ssoadm** attribute: `org-forgerock-auth-oauth-attribute-mapper`

## Attribute Mapper Configuration

Map of OAuth Provider user account attributes to local user profile attributes, with values in the form `provider-attr=local-attr`.

**ssoadm** attribute: `org-forgerock-auth-oauth-attribute-mapper-configuration`

## Save attributes in the session

When enabled, add the mapped attributes to the session saved. The default mode is `Enabled`.

**ssoadm** attribute: `org-forgerock-auth-oauth-save-attributes-to-session-flag`

## Email attribute in OAuth2 Response

Specifies the attribute identifying email address in the response from the profile service in the OAuth provider. This setting is used to send an email address with an activation code for accounts created dynamically.

**ssoadm** attribute: `org-forgerock-auth-oauth-mail-attribute`

## Create account if it does not exist

When enabled, if the user profile does not exist, optionally retrieve a password and activation code from the user, and then create the profile. The default mode is `Enabled`.

When the OAuth 2.0 client is configured to create new accounts, the SMTP settings must also be valid. As part of account creation, the OAuth 2.0 client authentication module sends the resource owner an email with an account activation code. To send the mail, OpenAM uses the SMTP settings you provide here in the OAuth 2.0 client configuration.

**ssoadm** attribute: `org-forgerock-auth-oauth-createaccount-flag`

## Prompt for password setting and activation code

When enabled, the user sets a password, receives an activation code by email. The user must correctly set both in order for the account to be created. The default mode is `Enabled`.

**ssoadm** attribute: `org-forgerock-auth-oauth-prompt-password-flag`

### Map to anonymous user

When enabled, map the OAuth authenticated user to the anonymous user you specify. No account is created, even if Create account if it does not exist is enabled.

**ssoadm** attribute: `org-forgerock-auth-oauth-map-to-anonymous-flag`

### Anonymous User

Specifies an anonymous user that exists in the current realm. The default is `anonymous`.

**ssoadm** attribute: `org-forgerock-auth-oauth-anonymous-user`

### OAuth 2.0 Provider logout service

Specifies the optional URL of the OAuth Provider.

**ssoadm** attribute: `org-forgerock-auth-oauth-logout-service-url`

### Logout options

Specifies whether not to log the user out without prompting from the OAuth Provider on logout, to log the user out without prompting, or to prompt the user regarding whether to logout from the OAuth provider.

**ssoadm** attribute: `org-forgerock-auth-oauth-logout-behaviour`

### Mail Server Gateway implementation class

Class to interact with the mail server. Default: `org.forgerock.openam.authentication.modules.oauth2.DefaultEmailGatewayImpl`

**ssoadm** attribute: `org-forgerock-auth-oauth-email-gwy-impl`

### SMTP host

Host name of the mail server. The default is `localhost`.

**ssoadm** attribute: `org-forgerock-auth-oauth-smtp-hostname`

### SMTP port

SMTP port number for the mail server. The default value is `25`.

**ssoadm** attribute: `org-forgerock-auth-oauth-smtp-port`

### SMTP User Name

If the mail server requires authentication to send mail, specifies the user name.

**ssoadm** attribute: `org-forgerock-auth-oauth-smtp-username`

### SMTP User Password

If the mail server requires authentication to send mail, specifies the password.

**ssoadm** attribute: `org-forgerock-auth-oauth-smtp-password`

### SMTP SSL Enabled

When enabled, connect to the mail server over SSL. OpenAM must be able to trust the SMTP server certificate.

**ssoadm** attribute: `org-forgerock-auth-oauth-smtp-ssl_enabled`

### SMTP From address

Specifies the message sender address, such as `no-reply@example.com`. The default value is `info@forgerock.com`.

**ssoadm** attribute: `org-forgerock-auth-oauth-smtp-email-from`

### Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `iplanet-am-auth-oauth-auth-level`

The following tables show endpoint URLs for OpenAM when configured as an OAuth 2.0 provider, and also URLs for large OAuth 2.0 providers. The default endpoints are for Facebook as the OAuth 2.0 provider.

In addition to the endpoint URLs you can set other fields, like scope and attribute mapping, depending on the provider you use.

*Table 2.1. Endpoint URLs for OpenAM*

OpenAM Field	Details
Authentication Endpoint URL	<code>/oauth2/authorize</code> under the deployment URL. <sup>a</sup> Example: <code>https://openam.example.com:8443/openam/oauth2/authorize</code> .
Access Token Endpoint URL	<code>/oauth2/access_token</code> under the deployment URL. <sup>a</sup> Example: <code>https://openam.example.com:8443/openam/oauth2/access_token</code> .
User Profile Service URL	<code>/oauth2/tokeninfo</code> under the deployment URL.

OpenAM Field	Details
	Example: <a href="https://openam.example.com:8443/openam/oauth2/tokeninfo">https://openam.example.com:8443/openam/oauth2/tokeninfo</a> .

<sup>a</sup>This OpenAM endpoint can take additional parameters. In particular you must specify the realm if the OpenAM OAuth 2.0 provider is configured for a subrealm rather than / (Top-Level Realm).

For example, if the OAuth 2.0 provider is configured for the realm `/customers`, then the authentication endpoint URL is as follows: <https://openam.example.com:8443/openam/oauth2/authorize?realm=/customers>

The `/oauth2/authorize` endpoint can also take `module` and `service` parameters. Use either as described in *Authenticating To OpenAM*, where `module` specifies the authentication module instance to use or `service` specifies the authentication chain to use when authenticating the resource owner.

*Table 2.2. Endpoint URLs for Facebook*

OpenAM Field	Details
Authentication Endpoint URL	<a href="https://www.facebook.com/dialog/oauth">https://www.facebook.com/dialog/oauth</a>
Access Token Endpoint URL	<a href="https://graph.facebook.com/oauth/access_token">https://graph.facebook.com/oauth/access_token</a>
User Profile Service URL	<a href="https://graph.facebook.com/me">https://graph.facebook.com/me</a>
OAuth 2.0 Provider logout service	<a href="http://www.facebook.com/logout.php">http://www.facebook.com/logout.php</a>

*Table 2.3. Endpoint URLs for Google*

OpenAM Field	Details
Authentication Endpoint URL	<a href="https://accounts.google.com/o/oauth2/auth">https://accounts.google.com/o/oauth2/auth</a>
Access Token Endpoint URL	<a href="https://accounts.google.com/o/oauth2/token">https://accounts.google.com/o/oauth2/token</a>
User Profile Service URL	<a href="https://www.googleapis.com/oauth2/v1/userinfo">https://www.googleapis.com/oauth2/v1/userinfo</a>
OAuth 2.0 Provider logout service	<a href="https://mail.google.com/mail/?logout">https://mail.google.com/mail/?logout</a>

*Table 2.4. Endpoint URLs for MSN*

OpenAM Field	Details
Authentication Endpoint URL	<a href="https://oauth.live.com/authorize">https://oauth.live.com/authorize</a>
Access Token Endpoint URL	<a href="https://oauth.live.com/token">https://oauth.live.com/token</a>
User Profile Service URL	<a href="https://apis.live.net/v5.0/me">https://apis.live.net/v5.0/me</a>
OAuth 2.0 Provider logout service	<a href="http://oauth.live.com/logout">http://oauth.live.com/logout</a>

## 2.2.17. Hints for the Persistent Cookie Module

The Persistent Cookie module supports configuration of cookie lifetimes, based on requests and a maximum time. It is the preferred method for creating a persistent cookie. If you choose this method, be aware that it does not interact with the persistent cookie options associated with the Core Authentication module.

**ssoadm** service name: `iPlanetAMAuthPersistentCookieService`

To configure the Persistent Cookie Module globally in the console, navigate to Configuration > Authentication > Persistent Cookie. In the window that appears you should see the following attributes:

### Idle Timeout

Specify the maximum idle time between requests, in hours. If that time is exceeded, the cookie is no longer valid.

**ssoadm** attribute: `openam-auth-persistent-cookie-idle-time`

### Max Life

Specify the maximum life of the cookie in hours.

**ssoadm** attribute: `openam-auth-persistent-cookie-max-life`

The Persistent Cookie module belongs with a second module in an authentication chain. To see how this works, navigate to Access Control > *Realm Name* > Authentication > New. Enter a name for the chain and add modules as shown in the figure. The following example shows how a Persistent Cookie module is sufficient; if that cookie does not yet exist, authentication in this case relies on a data store module such as LDAP.

**Persistent Cookie Chain - Properties** Save Reset Back to Authentication

(2 Item(s))

Add Remove Reorder

<input checked="" type="checkbox"/>	Instance	Criteria	Options
<input type="checkbox"/>	Pers_Cookie	SUFFICIENT	
<input type="checkbox"/>	LDAP	REQUIRED	

Scroll down in the properties page for the chain. You should set the `PersistentCookieAuthModule` as shown in the following figure:

Post Authentication Processing Class

Current Values `org.forgerock.openam.authentication.modules.persistentcookie.PersistentCookieAuthModule` Remove

New Value  Add

A list of post authentication processing classes for all users in this realm.

You should now be able to authenticate automatically, as long as the cookie exists for the associated domain.

#### Note

Unlike the legacy Core Authentication module, the Persistent Cookie module does not support the `iSPSCookie` option described in Section 2.5, "Authenticating To OpenAM"

## 2.2.18. Hints For the RADIUS Authentication Module

The Remote Authentication Dial-In User Service (RADIUS) module lets OpenAM authenticate users against RADIUS servers.

**ssoadm** service name: `iPlanetAMAuthRadiusService`

### Primary Radius Servers, Secondary Radius Servers

Specify the IP address or fully qualified domain name of the primary RADIUS server. The default is `127.0.0.1` (localhost loopback).

**ssoadm** attribute: `primary is iplanet-am-auth-radius-server1`; secondary is `iplanet-am-auth-radius-server2`

### Shared Secret

Specify the shared secret for RADIUS authentication. The shared secret should be as secure as a well-chosen password.

**ssoadm** attribute: `iplanet-am-auth-radius-secret`

### Port Number

Specify the RADIUS server port. Default is 1645.

**ssoadm** attribute: `iplanet-am-auth-radius-server-port`

### Timeout

Specify how many seconds to wait for the RADIUS server to respond. The default value is 3 seconds.

**ssoadm** attribute: `iplanet-am-auth-radius-timeout`

### Health check interval

Used for failover. Specify how often OpenAM performs a health check on a previously unavailable RADIUS server by sending an invalid authentication request. Default: 5 minutes

**ssoadm** attribute: `openam-auth-radius-healthcheck-interval`

### Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `iplanet-am-auth-radius-auth-level`

## 2.2.19. Hints For the SAE Authentication Module

The Secure Attribute Exchange (SAE) module lets OpenAM authenticate a user who has already authenticated with an entity that can vouch for the user to OpenAM, so that OpenAM creates a session for the user. This module is useful in virtual federation, where an existing entity instructs the local OpenAM instance to use federation protocols to transfer authentication and attribute information to a partner application.

**ssoadm** attribute: `sunAMAuthSAEService`

### Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `sunAMAuthSAEAuthLevel`

## 2.2.20. Hints For the SecurID Authentication Module

The SecurID module lets OpenAM authenticate users with RSA Authentication Manager software and RSA SecurID authenticators.

**ssoadm** service name: `iPlanetAMAuthSecurIDService`

### ACE/Server Configuration Path

Specify the directory in which the SecurID ACE/Server `sdconf.rec` file is located, which by default is expected under the configuration directory for OpenAM, such as `$HOME/openam/openam/auth/ace/data`. The directory must exist before OpenAM can use SecurID authentication.

**ssoadm** attribute: `iplanet-am-auth-securid-server-config-path`

### Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `iplanet-am-auth-securid-auth-level`

## 2.2.21. Hints For the Windows Desktop SSO Authentication Module

The Windows Desktop SSO module uses Kerberos authentication. The user presents a Kerberos token to OpenAM through the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) protocol. The Windows Desktop SSO authentication module enables desktop single sign on such that a user who has already authenticated with a Kerberos Key Distribution Center can authenticate to OpenAM without having to provide the login information again. Users might need to set up Integrated Windows Authentication in Internet Explorer to benefit from single sign on when logged on to a Windows desktop.

**ssoadm** service name: `iPlanetAMAuthWindowsDesktopSSOService`

### Service Principal

Specify the Kerberos principal for authentication in the following format.

```
HTTP/host.domain@dc-domain-name
```

Here, *host* and *domain* correspond to the host and domain names of the OpenAM instance, and *dc-domain-name* is the domain name of the Windows Kerberos domain controller server. The *dc-domain-name* can differ from the domain name for OpenAM.

You set up the account on the Windows domain controller, creating a computer account for OpenAM and associating the new account with a service provider name.

**ssoadm** attribute: `iplanet-am-auth-windowsdesktopsso-principal-name`

### Keytab File Name

Specify the full path of the keytab file for the Service Principal. You generate the keytab file using the Windows **ktpass** utility.

**ssoadm** attribute: `iplanet-am-auth-windowsdesktopsso-keytab-file`

### Kerberos Realm

Specify the Kerberos Key Distribution Center realm. For the Windows Kerberos service this is the domain controller server domain name.

**ssoadm** attribute: `iplanet-am-auth-windowsdesktopsso-kerberos-realm`

### Kerberos Server Name

Specify the fully qualified domain name of the Kerberos Key Distribution Center server, such as that of the domain controller server.

**ssoadm** attribute: `iplanet-am-auth-windowsdesktopsso-kdc`

### Return Principal with Domain Name

When enabled, OpenAM automatically returns the Kerberos principal with the domain controller's domain name during authentication.

**ssoadm** attribute: `iplanet-am-auth-windowsdesktopsso-returnRealm`

### Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `iplanet-am-auth-windowsdesktopsso-auth-level`

### Search for the user in the realm

Validates the user against the configured data stores. If the user from the Kerberos token is not found, authentication will fail. If an authentication chain is set, the user will be able to authenticate through another module.

**ssoadm** attribute: `iplanet-am-auth-windowsdesktopsso-lookupUserInRealm`

#### Note

Note: For Windows 7 and later, you will need to turn off the "Enable Integrated Windows Authentication" option in Internet Explorer. In addition, you will need to add and activate the `DisableNTMLPreAuth` key to the Windows Registry. For detailed instructions, see the Microsoft KB article on [when You cannot post data to a non-NTLM-authenticated Web site](#)

## 2.2.22. Hints For the Windows NT Authentication Module

The Windows NT module lets OpenAM authenticate against a Microsoft Windows NT server.

This module requires that you install a Samba client in a `bin` directory under the OpenAM configuration directory such as `$HOME/openam/openam/bin`.

**ssoadm** service name: `iPlanetAMAuthNTService`

### Authentication Domain

Specify the Windows domain name to which users belong.

**ssoadm** attribute: `iplanet-am-auth-nt-domain`

### Authentication Host

Specify the NetBIOS name of the Windows NT host to which to authenticate users.

**ssoadm** attribute: `iplanet-am-auth-nt-host`

### Samba Configuration File Name

Specify the full path to the Samba configuration file.

**ssoadm** attribute: `iplanet-am-auth-samba-config-file-name`

### Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `iplanet-am-auth-nt-auth-level`

## 2.2.23. Hints For the WSSAuth Authentication Module

The Web Service Security (WSSAuth) module lets OpenAM validate a user name, password combination received as an authentication token in a request from a Web Service Client to a Web Service Provider.

**ssoadm** service name: `sunAMAuthWSSAuthModuleService`

### User search attribute

Specify a user attribute to search for a user. Default is `uid`.

**ssoadm** attribute: `sunWebservicesUserSearchAttribute`

### User realm

Specify the realm to which users belong. For the OpenAM Security Token Service, this is `/`.

**ssoadm** attribute: `sunWebServicesUserRealm`

### User password attribute

Specify the password attribute or that of the password equivalent. The default is `userPassword`.

**ssoadm** attribute: `sunWebservicesUserPasswordAttribute`

### Authentication Level

Sets the authentication level used to indicate the level of security associated with the module. The value can range from 0 to any positive integer.

**ssoadm** attribute: `sunWebservicesAuthenticationLevel`

## 2.3. Configuring Authentication Chains

Once you have configured authentication modules, and added the modules to the list of module instances, you can configure authentication chains. Authentication chains let you handle situations where alternative modules are needed, or where a single set of credentials is not sufficient.

## Procedure 2.1. To Create an Authentication Chain

1. On the Access Control tab page of the OpenAM console, click the realm for which to create the authentication chain.
2. On the Authentication tab page for the realm, scroll to the bottom of the page, and click the New button in the Authentication Chaining table.
3. Give the new authentication chain a name, and add instances of the modules to use in the chain.
4. Assign appropriate criteria (optional, required, requisite, sufficient) as described above in Section 2.1, "About Authentication in OpenAM". You can also configure where OpenAM redirects the user upon successful and failed authentication, and plug in your post-authentication processing classes as necessary.
5. (Optional) If you need modules in the chain to share user credentials, then set options for the module.

### **iplanet-am-auth-shared-state-enabled**

Set `iplanet-am-auth-shared-state-enabled=true` to allow subsequent modules in the authentication chain to use the credentials, such as user name and password, captured by this module.  
(Default: `true`)

### **iplanet-am-auth-store-shared-state-enabled**

Set `iplanet-am-auth-store-shared-state-enabled=true` to store the captured credentials. Shared state is cleared when the user successfully authenticates, quits the chain, or logs out.  
(Default: `false`)

### **iplanet-am-auth-shared-state-behavior-pattern**

Set `iplanet-am-auth-shared-state-behavior-pattern=tryFirstPass` (the default) to try authenticating with the captured password. If authentication fails, then OpenAM prompts the user for the credentials again.

Set `iplanet-am-auth-shared-state-behavior-pattern=useFirstPass` to authenticate with the captured password. If authentication fails, then the module fails.

For example, consider a chain with two modules sharing credentials according to the default settings. The first module in the chain has the option `iplanet-am-auth-shared-state-enabled=true`, and criteria `REQUIRED`. The second module in the chain has options `iplanet-am-auth-shared-state-enabled=true`, `iplanet-am-auth-shared-state-behavior-pattern=tryFirstPass`, and criteria `REQUIRED`. A successful authentication sequence happens as follows. The user enters her credentials for the first module, successfully authenticating. The first module shares the credentials with the second module, successfully authenticating the user without prompting again for her credentials, unless the credentials for the first module do not successfully authenticate here to the second module. Just be sure to separate the options with a space and not a comma.

6. Save your work.

### Procedure 2.2. To Select the Default Chain

Before you select the default chain for users, and especially for administrators, test the authentication chain first. For example, <http://openam.example.com:8080/openam/UI/Login?service=NewChain>. If you cannot log in, then go back and fix the authentication chain's configuration before making it the default.

1. On the Access Control tab page of the OpenAM console, click the realm for which to set the default authentication chain.
2. (Optional) If necessary, on the Authentication tab page for the realm, adjust the drop-down lists for Organization Authentication Configuration and Administrator Authentication Configuration to the appropriate authentication chains.

The Organization Authentication Configuration serves when users access </openam/UI/Login>.

The Administrator Authentication Configuration serves when users access </openam/console>.

You can set these independently to separate administrative login from user login. For example, you can change the default user chain, but leave the default administrator chain as is to avoid locking yourself out as administrator. By default, `amadmin` can login at </openam/UI/Login>. You can change that for your deployment.

3. Save your work.

## 2.4. Post Authentication Plugins

Post authentication plugins include custom processing at the end of the authentication process, immediately before the subject is authenticated. Common uses of post authentication plugins include setting cookies and session variables. Post authentication plugins are often used in conjunction with policy agents. The post authentication plugin sets custom session properties, and then the policy agent injects the custom properties into the request header to the protected application.

In the OpenAM console, you add post authentication plugins to an authentication chain. Navigate to Access Control > *Realm Name* > Authentication > Authentication Chaining > *Auth Chain Name*. Scroll down to the Post Authentication Processing Class list.

### Standard Post Authentication Plugins

OpenAM provides some post authentication plugins as part of the standard product delivery.

**Class name:** `org.forgerock.openam.authentication.modules.adaptive.Adaptive`

The adaptive authentication plugin serves to save cookies and profile attributes after successful authentication.

Add it to your authentication chains that use the adaptive authentication module configured to save cookies and profile attributes.

**Class name:** `org.forgerock.openam.authentication.modules.oauth2.OAuth2PostAuthnPlugin`

The OAuth 2.0 post authentication plugin builds a global logout URL used by `/oauth2c/OAuthLogout.jsp` after successful OAuth 2.0 client authentication. This logs the resource owner out with the OAuth 2.0 provider when logging out of OpenAM.

Before using this plugin, configure the OAuth 2.0 authentication module with the correct OAuth 2.0 Provider logout service URL, and set the Logout options to Log out or Prompt. This plugin cannot succeed unless those parameters are correctly set.

Sometimes OAuth 2.0 providers change their endpoints, including their logout URLs. When using a provider like Facebook, Google, or MSN make sure you are aware when they change their endpoint locations so that you can change your client configuration accordingly.

**Class name:** `org.forgerock.openam.authentication.plugins.AccountExpirePlugin`

The account expiration post authentication plugin sets an account expiration date after successful authentication. OpenAM uses this to prevent expired accounts from being used to authenticate.

The default of 30 days can be changed using the advanced OpenAM server property, `org.forgerock.openam.authentication.accountExpire.days`.

If necessary, you can also write your own custom post authentication plugin as described in the *Developer's Guide* chapter on *Creating a Post Authentication Plugin* in the *Developer's Guide*.

## 2.5. Authenticating To OpenAM

This section explains how to connect to OpenAM for user authentication by adding parameters to the login URL when testing your configuration.

The base URL to authenticate to OpenAM points to `/UI/Login` under the deployment URL, such as `http://openam.example.com:8080/openam/UI/Login`.<sup>2</sup> You can, however, specify parameters in the query string of the URL to request a specific authentication configuration. For example, `http://openam.example.com:8080/openam/UI/Login?module=LDAP` requests that OpenAM use the LDAP authentication module.

OpenAM accepts the following parameters in the query string. With the exception of `IDToken` parameters, use no more than one occurrence of each.

### Note

The way you enter parameters depends on whether you are using the classic UI or the XUI. For example, if you want to request that OpenAM end the user's current session and start a new session in the classic UI, you

<sup>2</sup>The base URL to logout is similar, for example `http://openam.example.com:8080/openam/UI/Logout`.

might enter a URL similar to: `http://openam.example.com:8080/openam/UI/Login?locale=fr`. Alternatively, for the JavaScript-based XUI, the corresponding URL would be subtly different: `http://openam.example.com:8080/openam/XUI/#login/&arg=newsession`.

### **arg=newsession**

Request that OpenAM end the user's current session and start a new session.

### **authlevel**

Request that OpenAM authenticate the user using a module with at least the specified authentication level that you have configured.

As this parameter determines authentication module selection, do not use it with `module`, `service`, or `user`.

### **ForceAuth**

If `ForceAuth=true`, request that OpenAM force the user to authenticate even if she already has a valid session. On successful authentication, OpenAM updates the session token.

### **goto**

On successful authentication, or successful logout, request that OpenAM redirect the user to the specified location. Values must be URL encoded.

### **gotoOnFail**

On authentication failure, request that OpenAM redirect the user to the specified location. Values must be URL encoded.

### **IDToken1, IDToken2, ..., IDTokenN**

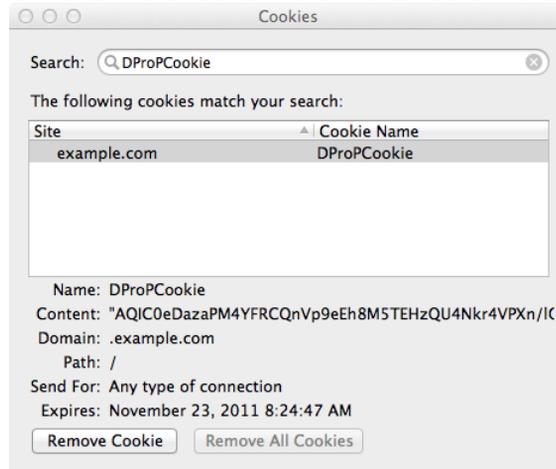
Pass the specified credentials as `IDToken` parameters in the URL. The `IDToken` credentials map to the fields in the login page for the authentication module, such as `IDToken1` as user ID and `IDToken2` as password for basic user name, password authentication. The order depends on the callbacks in login page for the module; `IDTokenN` represents the N<sup>th</sup> callback of the login page.

### **iPSPCookie=yes**

Applicable only if you configure persistent cookies based on Section 2.2.5.3, "Core - Persistent Cookie (Legacy)"

Request that OpenAM return a persistent cookie that remains in the browser after the browser is closed, allowing the user to login again without being prompted for credentials. This only works if you have configured persistent cookie mode for the realm where the user logs in.

OpenAM sets an `DProPCookie` that persists until expiry. The following screen shot shows an example.



An alternative persistent cookie mechanism extends the lifetime of the normal `iPlanetDirectoryPro` using the advanced server settings, `openam.session.persist_am_cookie` or `openam.session.allow_persist_am_cookie`, and `com.iplanet.am.cookie.timeToLive`.

To set the mechanism globally for the server, browse in the OpenAM console to Configuration > Servers and Sites > *Server Name* > Advanced, and then set `openam.session.persist_am_cookie` to `true` and `com.iplanet.am.cookie.timeToLive` to the cookie lifetime in seconds.

To allow users to use this mechanism on a per-session basis, browse in the OpenAM console to Configuration > Servers and Sites > *Server Name* > Advanced, and then set `openam.session.allow_persist_am_cookie` to `true` and `com.iplanet.am.cookie.timeToLive` to the cookie lifetime in seconds. (If the OpenAM .war deployed does not include the console, set these properties in the .properties configuration file.) Also configure the session properties either globally under Configuration > Global > Session > Dynamic Attributes, or per realm under Access Control > *Realm Name* > Services > Session. Then, to request the cookie, use `openam.session.persist_am_cookie=Yes` as one of the query string parameters in the login URL.

#### Note

Neither the XUI nor the Persistent Cookie module supports the deprecated `iSPCookie`.

## locale

Request that OpenAM display the user interface in the specified, supported locale. Locale can also be set in the user's profile, in the HTTP header from her browser, configured in OpenAM, and so on.

## module

Request that OpenAM use the authentication module instance as configured for the realm where the user is authenticating.

As this parameter determines authentication module selection, do not use it with `authlevel`, `service`, or `user`.

### realm

Request that OpenAM authenticate the user to the specified realm.

### service

Request that OpenAM authenticate the user with the specified authentication chain.

As this parameter determines authentication module selection, do not use it with `authlevel`, `module`, or `user`.

### user

Request that the user, specified by her OpenAM universal ID, authenticate according to the chain specified in her profile.

As this parameter determines authentication module selection, do not use it with `authlevel`, `module`, or `service`.

## 2.6. Configuring Account Lockout

OpenAM supports two different approaches to *account lockout*, where OpenAM locks an account after repeated authentication failures. Lockout works with modules for which users can enter a password incorrectly.

- Memory lockout locks the user account, keeping track of the locked state only in memory, and then unlocking the account after a specified delay. Memory lockout is also released when OpenAM restarts.
- Persistent (physical) lockout sets the user account status to `inactive` in the user profile. For persistent lockout, OpenAM tracks failed authentication attempts by writing to the user repository.

Persistent account lockout works independently of account lockout mechanisms in the underlying directory server that serves as the user data store.

You configure account lockout by editing settings for the core authentication module. Access the settings in OpenAM console under Access Control > *Realm Name* > Authentication > All Core Settings..., and then scroll down to the Account Lockout section. The inline help explains the settings in detail.

- Enable lockout by checking Login Failure Lockout Mode, setting the number of attempts, and setting the lockout interval and duration.

You can also opt to warn users after several consecutive failures, or to multiply the lockout duration on each successive lockout.

- You can set up email notification upon lockout to an administrator if OpenAM is configured to send mail. (Configuration > Servers and Sites > Default Server Settings > General > Mail Server.)
- For persistent lockout, OpenAM sets the value of the user's `inetuserstatus` profile attribute to `inactive`. You can also specify another attribute to update on lockout. You can further set a non-default attribute on which to store the number of failed authentication attempts. When you do store the number of failed attempts in the data store, other OpenAM servers accessing the user data store can also see the number.

If you need to unlock a user's account, find the user under Access Control > *Realm Name* > Subjects > User, set the user's User Status to Active, and click Save.

## 2.7. Configuring Session Quotas

OpenAM lets you limit the number of active sessions for a user by setting session quotas. You also configure session quota exhaustion actions so that when a user goes beyond the session quota, OpenAM takes the appropriate action.

### Important

To enforce session quotas across multiple servers in a site, configure session failover as described in the *Installation Guide* chapter, *Setting Up OpenAM Session Failover in the Installation Guide*.

### Procedure 2.3. To Configure Session Quotas & Exhaustion Actions

The session quota applies to all sessions opened for the same user (as represented by the user's universal identifier).

1. Log in to OpenAM Console as administrator, and then browse to Configuration > Global > Session.
2. Set Enable Quota Constraints to **ON**.
3. Set Resulting behavior if session quota exhausted.

The following settings are available by default.

#### **DENY\_ACCESS**

Deny access, preventing the user from creating an additional session.

#### **DESTROY\_NEXT\_EXPIRING**

Remove the next session to expire, and create a new session for the user. The next session to expire is the session with the minimum time left until expiration.

This is the default setting.

### DESTROY\_OLDEST\_SESSION

Remove the oldest session, and create a new session for the user.

### DESTROY\_OLD\_SESSIONS

Remove all existing sessions, and create a new session for the user.

If none of these session quota exhaustion actions fit your deployment, you can implement a custom session quota exhaustion action. See the *Developer's Guide* chapter on *Customizing Session Quota Exhaustion Actions* in the *Developer's Guide* for an example.

#### 4. Set Active User Sessions to the session quota.

The default is 5 sessions.

#### 5. Save your work.

#### 6. (Optional) If you have multiple servers but session failover is not configured, configure multi-server mode as described below.

- If you have only a single OpenAM server, skip this step. OpenAM enforces the session quota you set for the server.
- If you have multiple servers with session failover configured, then also skip this step. In this case OpenAM uses the session store to enforce session quotas globally across your deployment. In other words when the Set Active User Sessions is 5, a user can have a maximum of 5 active sessions.
- If you have multiple OpenAM servers but session failover is not configured, configure multi-server mode for session quotas. Browse to Configuration > Servers and Sites > Default Server Settings or Configuration > Servers and Sites > *Server Name*, and then use the Advanced tab page to set the following advanced server property.

```
openam.session.useLocalSessionsInMultiServerMode = true
```

When you set this property to `true` for your OpenAM servers, users can potentially reach the session quota for each individual server before all session quotas are exhausted. In other words if you have 4 OpenAM servers and Set Active User Sessions is 5, then the user can have a maximum of 20 (5 \* 4) sessions.

## Chapter 3

# Defining Authorization Policies

*Authorization* is determining whether to grant or deny a user access to a resource. *Policies* define how to determine whether to grant or deny access. This chapter describes how to configure authorization policies managed by OpenAM.

## 3.1. About Authorization in OpenAM

Applications rely on OpenAM for access management, which breaks down into authentication, or determining who is trying to access a resource, and authorization, or determining whether to grant or deny access. This is because whether access is granted generally depends on what the rules about access are, who is trying to gain access, and perhaps some other conditions, such as whether the access itself needs to happen over a secure channel or what time of day it is. To return to the international airport example, the rule may be that passengers with valid passports and visa presenting valid plane tickets are allowed through to the gate where the plane is waiting to take off, but only under the condition that the plane is going to leave soon. (You cannot expect to get to the gate today with a scheduled departure for three months from now.)

To allow OpenAM to determine whether to grant access, you define authorization policies. A policy includes *rules* that match what resources a user aims to access in what way and whether to grant or deny that access, *subjects* to whom the policy applies, and potentially *conditions* under which the policy applies. When queried about whether to let a user through to a protected resource, OpenAM decides to authorize access or not based on the applicable policy. OpenAM then communicates its decision to the application using OpenAM for access management, or in the common case to the policy agent installed in the server where the application runs. The application or the agent then enforces the authorization decision from OpenAM.

For example, consider the case where OpenAM protects a web site page that users access to update their profiles. An OpenAM policy agent installed in the web server intercepts client requests to enforce policy. The policy says that authenticated users can access the page to update their profiles as long as they come in over HTTPS, rather than HTTP.

When a user browses to the page to update her profile, the OpenAM policy agent intercepts the request. The policy agent notices that the request is to access a protected resource, but the request is coming from a user who has not yet logged in and has no authorization to visit the page. The agent therefore redirects the user to OpenAM.

OpenAM receives the redirected user, serving a login page that collects her email and password. With the email and password credentials, OpenAM authenticates the user, and gives her a session. OpenAM then redirects the user to the policy agent, which gets the policy decision from OpenAM

for the page she wants to access, and grants access to the page. OpenAM and the policy agent use cookies set in the user's browser to reference her session. While the user has a valid session with OpenAM, she can go away to another page in her browser, come back to the update profile page, and gain access without having to enter her email and password again.

Notice how OpenAM and the policy agent handle the access in the example. The web site developer can offer a profile page, but the web site developer never has to manage login, nor has to handle who can access a page. As OpenAM administrator, you can change authentication and authorization independently of updates to the web site. You might need to agree with web site developers on how OpenAM identifies users so web developers can find their particular profiles, or identify the user by her own name when she logs in. Yet your organization is now spared from new web site development projects when you want to add external access to your Intranet for roaming users, open certain of your sites to partners, only let managers access certain pages of your HR web site, or allow users already logged in to their desktops to visit protected web sites without having to type their user names and passwords again.

When OpenAM denies a request due to a condition that could be corrected by further authentication, OpenAM can send advice to the policy agent, and the policy agent can then take remedial action. For instance, suppose a user comes to a web site having authenticated with an email address and password, which is configured as authentication level 0. Had the user authenticated over the VPN which relies on one-time password authentication, she would have had authentication level 1 in her session. Yet, because she has authentication level 0, she currently cannot access the page she wants, which requires authentication level 1. OpenAM can send advice in this case, prompting the agent or application to redirect her to authenticate again with a one-time password, gaining authentication level 1, and thus having OpenAM grant her access to the protected page.

Policies can include *response providers*. Response providers extend HTTP headers with additional information beyond an "allow" or "deny" decision. For example, a response provider can return a message about why access was denied.

## 3.2. Configuring Policies

An OpenAM authorization policy defines who can access what, under what conditions. The OpenAM agents enforcing policy call upon OpenAM to make policy decisions. Decisions from OpenAM can be as simple as "allow" or "deny." Decisions from OpenAM can alternatively provide additional information required for policy enforcement. OpenAM policies use response providers to return such additional information.

### *Procedure 3.1. To Create a Policy*

1. In the OpenAM console, select Access Control > *Realm Name* > Policies, then click New Policy.
2. Provide at minimum a name for the policy.

### *Procedure 3.2. To Configure a Policy For a Web Site*

Once a policy is created, you can further specify rules, subjects, conditions, and response providers.

OpenAM has, by default, three kinds of resources that you can protect with a policy.

- A *Discovery Service*, used in federated access management, locates the web service providing the data needed to complete an operation. Your policy protects what clients can look up and what they can update.
- A *Liberty Personal Profile Service*, used in federated access management, provides an identity's basic profile information. Your policy protects what clients can query and what they can modify.
- A *URL Policy Agent* protects resources on a specific web site or web application. Your policy protects what URLs client applications can access with HTTP GET and POST operations.

Follow these steps to configure a policy to protect a web site or web application.

1. In the OpenAM console, select Access Control > *Realm Name* > Policies > *Policy Name* to display the policy to edit.
2. In the Rules table, click New, and then create at least one Rule, identifying a resource name to protect.

A Rule can specify an individual URL or resource name to protect. Alternatively, a Rule can match URLs or resource names by using wildcards.

- The wildcard `*` matches multiple levels in a path. When used at the end of a rule `*` matches one or more characters, not zero or more characters.

For example, `http://www.example.com/*` matches `http://www.example.com/index.html` and also `http://www.example.com/company/images/logo.png`, but not `http://www.example.com/`.

- The wildcard `-*-` matches a single level in a path.

For example, `http://www.example.com/-*-` matches `http://www.example.com/index.html` but does not match `http://www.example.com/company/images/logo.png`.

- Wildcards do not match `?`. You must explicitly add Rules to match URLs with query strings.

For example, to match everything under `http://www.example.com/` specify three Rules, one for `http://www.example.com/`, one for `http://www.example.com/*`, and one for `http://www.example.com/?*`.

When defining Rules to match URLs with query strings, know that OpenAM sorts the query string field-value pairs by field name when normalizing URLs, before checking whether a policy matches. Therefore the query string `?subject=SPBnfm+t5PlP+ISyQhVlpLE22A8=&action=get` is equivalent to the query string `?action=get&subject=SPBnfm+t5PlP+ISyQhVlpLE22A8=`.

- Duplicate and trailing slashes (`/`) are not considered part of the resource name to match.

For example, `http://www.example.com/path`, `http://www.example.com/path/`, `http://www.example.com//path/`, and `http://www.example.com/path//` are all treated in the same way.

- Wildcards can be used in protocols, host names, and port numbers.

For example, `http*://*:*/*` matches `http://www.example.com/index.html`, `https://www.example.com/index.html`, and `http://www.example.net:8080/index.html`.

When a port number is not explicitly specified, then the default port number is implied. Therefore `http://www.example.com/*` is the same as `http://www.example.com:80/*`, and `https://www.example.com/*` is the same as `https://www.example.com:443/*`.

- Wildcards cannot be escaped.
- Do not mix `*` and `-*` in the same Rule.
- By default, comparisons are not case sensitive. The delimiter, wildcards and case sensitivity are configurable. To see examples of other configurations, browse in the OpenAM Console to Configuration > Global > Policy Configuration > Resource Comparator.

Follow these steps for each new Rule you create.

- a. Select URL Policy Agent (with resource name), and click Next.
- b. Name the new rule, add the URL to protect in the Resource Name field, and set whether to allow or deny HTTP GET and POST requests to the URL.
- c. Click Finish.

The new rule is not yet saved until you click the Save button in the Edit Policy screen.

3. In the Subjects table, click New... to define a subject, identifying the users to whom the policy applies.
  - a. In the Select Subject Type screen, make your selection, and then click next.
    - Authenticated Users refers to users who have authenticated with OpenAM, even if they do not have profiles in the realm where you define the policy.
    - OpenAM Identity Subject refers to users or groups you can find under Access Control > Realm Name > Subjects.
    - Web Services Clients are for federated access management.
  - b. Name the subject.
  - c. (Optional) If you want to apply the policy to everyone but the subjects you identified, then select Exclusive.

For example

- d. (Optional) If you selected OpenAM Identity Subject, use the Filter section to find and add to your list the subjects to whom to apply the policy.
  - e. Click Finish.
4. (Optional) In the Conditions table, click New... to create a condition, constraining the circumstances under which the policy applies.
- a. Select the Condition Type from the list.
    - Active Session Time lets you make the policy depend on how long the user's session has been active, and even to terminate the session if deemed too old, such that the user must authenticate again.
    - Authentication by Module Chain lets you make the policy depend on the realm where the user authenticated, and on the authentication chain used to authenticate.
    - Authentication by Module Instance lets you make the policy depend on the realm where the user authenticated, and on the authentication module used to authenticate, as well as setting timeouts for application authentication.
    - Authentication Level (greater than or equal to) lets you make the policy depend on the realm where the user authenticated, and on a minimum acceptable authentication level.
    - Authentication Level (less than or equal to) lets you make the policy depend on the realm where the user authenticated, and on a maximum acceptable authentication level.
    - Authentication to a Realm lets you make the policy depend on the realm where the user authenticated.
    - Current Session Properties lets you make the policy depend on attributes set in the user's session.
    - Identity Membership lets you make the policy depend on a list of OpenAM subjects that you select, and whether the user belongs to the list of users or is a member of a group you selected.
    - IP Address/DNS Name lets you apply the policy to clients in specific IP address ranges or coming from a particular DNS domain. It also lets you choose between IPv4 and IPv6 address versions.
      - For IPv4, fill in the fields for the From and To with four sets of four characters.
      - For IPv6, fill in the fields for the From and To with eight sets of four hexadecimal characters. The default separator between each set of IPv6 hexadecimal characters is a colon.

- For a hybrid address, select IPv6. Fill in the first four fields for the From and To with four characters of the IPv6 address, then change the punctuation to a . and add the four IPv4 characters in the remaining three fields.
- LDAP Filter Condition lets you make the policy depend on whether the user's entry can be found using the LDAP search filter you specify in the directory configured for the policy service, which by default is the identity repository. See Configuration > Global > Policy Configuration > Realm Attributes > Primary LDAP Server.

Alternatively you can set this for the realm under Access Control > *Realm Name* > Services > Policy Configuration.

- Resource/Environment/IP Address lets you make the policy apply using a complex condition such as whether the user is making a request from the localhost and has authenticated with the LDAP authentication module. This address also works with IPv4, IPv6, or a hybrid of the two.
  - Time (day, date, time, and timezone) lets you make the policy depend on when the policy is evaluated.
- a. Based on the Condition Type you choose, configure the condition, and then click Finish.
5. (Optional) In the Response Providers table, click New to set up a response provider that adds attributes retrieved from the user entry in the identity repository into the headers of the request at policy decision time.
    - a. Name the provider.
    - b. Add static attributes having the form *attribute=value*.
    - c. Add dynamic attributes having the form *responseAttr=repoAttr*, where *responseAttr* is the attribute name to be put into the header of the request, and *repoAttr* is the attribute name used in the identity repository.
    - d. Click Finish.
  6. Save your work.

### 3.3. How OpenAM Reaches Policy Decisions

OpenAM has to match policies to resources to take policy decisions. For a policy to match, the resource has to match a resources identified in a rule. The user making the request has to match a subject. Furthermore, at least one condition for each condition type has to be satisfied.

If more than one policy matches, OpenAM has to reconcile differences. When multiple policies match, the order in which OpenAM uses them to make a policy decision is not deterministic. However, a deny decision overrides an allow decision, and so by default once OpenAM reaches a deny decision

its stops checking further policies. (If you want OpenAM to continue checking despite the deny, see Configuration > Global > Policy Configuration > Continue Evaluation on Deny Decision.)

## 3.4. Managing Policies Outside the Console

When you first create policies, the OpenAM console helps you to get started quickly. Yet, when you have many policies to manage you might find it easier to script operations, starting from policies originally created in the console, then exported to XML.

### *Procedure 3.3. To Export Policies From the Console*

You can export policies created in the console to an XML Policies document.

- Use the **ssoadm list-policies** command.

```
$ ssoadm
  list-policies
  --realm "/"
  --adminid amadmin
  --password-file /tmp/pwd.txt

Policy definitions were returned under realm, /.
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Policies
PUBLIC "-//OpenSSO Policy Administration DTD//EN"
"jar://com/sun/identity/policy/policyAdmin.dtd">

<!-- extracted from realm, / -->
<Policies>
<Policy name="URL Policy" createdby="id=amadmin,ou=user,o=openam"
  lastmodifiedby="id=amadmin,ou=user,o=openam" creationdate="1312553988059"
  lastmodifieddate="1315403023466" referralPolicy="false" active="true" >
<Rule name="Allow GET with parameters">
<ServiceName name="iPlanetAMWebAgentService" />
<ResourceName name="http://www.example.com/ching/*?*"/>
<AttributeValuePair>
<Attribute name="GET" />
<Value>allow</Value>
</AttributeValuePair>
</Rule>
<Rule name="Allow GET and POST">
<ServiceName name="iPlanetAMWebAgentService" />
<ResourceName name="http://www.example.com/ching/*" />
<AttributeValuePair>
<Attribute name="POST" />
<Value>allow</Value>
</AttributeValuePair>
<AttributeValuePair>
<Attribute name="GET" />
<Value>allow</Value>
</AttributeValuePair>
</Rule>
<Subjects name="Subjects:1312553593870WmIuFvI=" description="">
<Subject name="All Authenticated Users" type="AuthenticatedUsers"
```

```
includeType="inclusive">
</Subject>
</Subjects>
</Policy>
</Policies>
```

### Procedure 3.4. To Import Policies Using the Command Line

In a production environment where you manage operations using scripts rather than the console, use exported, file-based policies edited for your needs, and then import the policies using **ssoadm**.

#### 1. Create your XML policy file.

```
$ cat policy.xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Policies
PUBLIC "-//OpenSSO Policy Administration DTD//EN"
"jar://com/sun/identity/policy/policyAdmin.dtd">
<!-- New policy, same as the old policy -->
<Policies>
<Policy name="New Policy" referralPolicy="false" active="true" >
<Rule name="Allow GET with parameters">
<ServiceName name="iPlanetAMWebAgentService" />
<ResourceName name="http://www.example.com/ching/*?*"/>
<AttributeValuePair>
<Attribute name="GET" />
<Value>allow</Value>
</AttributeValuePair>
</Rule>
<Rule name="Allow GET and POST">
<ServiceName name="iPlanetAMWebAgentService" />
<ResourceName name="http://www.example.com/ching/*" />
<AttributeValuePair>
<Attribute name="POST" />
<Value>allow</Value>
</AttributeValuePair>
<AttributeValuePair>
<Attribute name="GET" />
<Value>allow</Value>
</AttributeValuePair>
</Rule>
<Subjects name="Subjects" description="Everybody authenticated">
<Subject name="All Authenticated Users" type="AuthenticatedUsers"
includeType="inclusive">
</Subject>
</Subjects>
</Policy>
</Policies>
```

#### 2. Use the **ssoadm create-policies** command.

```
$ ssoadm  
create-policies  
--realm "/"  
--adminid amadmin  
--password-file /tmp/pwd.txt  
--xmlfile policy.xml  
  
Policies were created under realm, /.
```

## 3.5. Delegating Policy Management & Decisions

You use a *referral* to delegate policy management, and to delegate policy decision making.

Referrals are covered in the chapter on Realms.

## Chapter 4

# Defining Entitlements

This chapter covers how to define entitlements for fine-grained authorization to access particular resources.

## 4.1. About Entitlements

OpenAM *entitlements* serve much the same purpose as OpenAM policies, defining who has access to what, under what conditions. OpenAM stores and manages policies centrally using the standard eXtensible Access Control Markup Language (XACML). You can access OpenAM entitlements and policy decisions using the RESTful web interface, for even lighter weight policy enforcement than with OpenAM policy agents.

The OpenAM entitlements service uses XACML terminology to refer to the different points dealing with policy.

- OpenAM serves as a *policy administration point* (PAP) where you define, store, and manage policies. OpenAM uses the configuration directory to store entitlements, whereas profiles are stored in the identity repository (user data store).
- OpenAM also serves as a *policy decision point* (PDP), evaluating policies and issuing authorization decisions, and as a *policy information point*, providing the information needed for authorization decisions.
- OpenAM policy agents act as *policy enforcement points*, obtaining decisions from PDPs to protect access to resources.

Entitlement policies define who has who has access to what, under what conditions, in the same way that other OpenAM policies define policy. Entitlement policies do let you define virtual subjects and subjects based on attribute lookup to determine who has access to the resources.

Entitlements apply for *applications*, which in this context mean protected resources that share a common set of actions and related policies. For example, the web agent application protects web resources accessed through HTTP GET and POST actions using a web policy agent to enforce decisions to allow or deny access. You can also define more specific applications as demonstrated by the examples delivered with OpenAM.

*Delegations* grant specific users privileges to manage policies.

## 4.2. Managing Entitlements on the Command Line

To manage entitlements, you can use the **ssoadm** command. The **ssoadm** command provides several other subcommands for managing entitlements in addition to those shown here.

### Procedure 4.1. To List Current Policies

- Use the **ssoadm list-xacml** command to list current policies, ready for export.

```
$ ssoadm list-xacml --realm / --adminid amadmin --password-file /tmp/pwd.txt

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PolicySet PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:
rule-combining-algorithm:deny-overrides" Version="2011.10.07.12.22.04.705"
PolicySetId="/:2011.10.07.12.22.04.704" xmlns="urn:oasis:names:tc:xacml:3.0:
core:schema:cd-1">
  <Target/>
  ... other policies ...
  <Policy RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:
rule-combining-algorithm:deny-overrides" Version="2011.10.07.11.51.27.444"
PolicyId="New Policy">
  ... policy content here ...
  </Policy>
</PolicySet>
```

### Procedure 4.2. To Import an Entitlements Policy

- Use the **ssoadm create-xacml** command to import a policy.

```
$ ssoadm
create-xacml
--realm /
--xmlfile policy.xml
--adminid amadmin
--password-file /tmp/pwd.txt
```

### Procedure 4.3. To Create an Application

- Use the **ssoadm create-appl** command to create an application type.

```
$ cat application.txt
resources=http://myapp.example.com:80/*
subjects=com.sun.identity.admin.model.IdRepoUserViewSubject
subjects=com.sun.identity.admin.model.VirtualViewSubject
subjects=com.sun.identity.admin.model.OrViewSubject
subjects=com.sun.identity.admin.model.AndViewSubject
conditions=com.sun.identity.admin.model.DateRangeCondition
conditions=com.sun.identity.admin.model.DaysOfWeekCondition
conditions=com.sun.identity.admin.model.IpRangeViewCondition
conditions=com.sun.identity.admin.model.DnsNameViewCondition
conditions=com.sun.identity.admin.model.TimeRangeCondition
conditions=com.sun.identity.admin.model.TimezoneCondition
conditions=com.sun.identity.admin.model.OrViewCondition
conditions=com.sun.identity.admin.model.AndViewCondition
conditions=com.sun.identity.admin.model.NotViewCondition
entitlementCombiner=com.sun.identity.entitlement.DenyOverride
$ ssoadm
  create-appl
  --realm /
  --applicationtype iPlanetAMWebAgentService
  --name myApp
  --adminid amadmin
  --password-file /tmp/pwd.txt
  --datafile application.txt

myApp was created.
```

## Chapter 5

# Configuring Realms

This chapter shows how to configure OpenAM *realms*, which are used to group configuration and identities together. For example, you might have one realm for OpenAM administrators and agents, and another realm for users. In this two-realm setup, the OpenAM administrator can login to the administrative realm to manage the services, but cannot authenticate as OpenAM administrator to the realm that protects web sites with HR and financial information.

OpenAM associates a realm with at least one identity repository and authentication process. OpenAM also associates the realm with authorization policies and entitlements for users, and privileges for administrators. Each realm can have its own configuration for services.

When you first configure OpenAM, OpenAM sets up the default / (Top Level Realm), containing OpenAM configuration data, and allowing authentication using the identity repository you choose during initial configuration. The top level realm might hold the overall configuration for Example.com for instance.

You create new realms to subdivide authentication, and authorization, and to delegate management of sub-realms. For example, your organization might require separate realms for payroll, human resources, and IT management domains and their applications.

- Procedure 5.1, "To Create a New Realm"
- Procedure 5.2, "To Delegate Administration"
- Procedure 5.3, "To Delegate Policy Management"
- Procedure 5.4, "To Configure a Data Store for a Realm"
- Procedure 5.5, "To Configure a Web or J2EE Agent For Login to a Realm"

### *Procedure 5.1. To Create a New Realm*

You can create a new realm through the OpenAM console as described below, or by using the **ssoadm create-realm** command.

1. Login to the OpenAM console as OpenAM Administrator, **amadmin**.
2. On the Access Control tab > Realms table, click New... to open the New Realm page, where you configure the realm.

If you configure the realm to be inactive, then users cannot use it to authenticate or be granted access to protected resources.

Realm/DNS aliases must follow standard FQDN conventions, such as `hr.example.com` or `pay.example.com`.

3. Save your work after defining the configuration for the new realm.

You configure a realm through the console starting from the Access Control tab > Realms table. By default the new realm inherits configuration from the global configuration. The default identity repository is the one you choose when configuring OpenAM after deployment. The default authentication mechanism corresponds to that identity repository as well. You can, however, constrain authentication to rely on different data stores, and set policy for agents to define authorization in the realm.

### *Procedure 5.2. To Delegate Administration*

You can delegate administration in a realm. OpenAM grants administrative capabilities to members of groups having administrative privileges.

You can grant privileges through the OpenAM console as described below, or by using the **ssoadm add-privileges** command.

1. On the Access Control tab > Realms table, click the realm for which you want to delegate administration to view the realm configuration.
2. On the Privileges tab, click the name of the group to whom you intend to grant access.
3. Select the administrative privileges to delegate for the realm, and then save your work.

When using the **ssoadm add-privileges** command, the privilege names are as follow.

#### **AgentAdmin**

Read and write access to all configured Agents

#### **EntitlementRestAccess**

REST calls for Policy Evaluation

#### **FederationAdmin**

Read and write access to all federation metadata configurations

#### **LogAdmin**

Read and write access to all log files

**LogRead**

Read access to all log files

**LogWrite**

Write access to all log files

**PolicyAdmin**

Read and write access only for policy properties

**PrivilegeRestAccess**

REST calls for managing entitlements

**PrivilegeRestReadAccess**

REST calls for searching entitlements

**RealmAdmin**

Read and write access to all realm and policy properties

### *Procedure 5.3. To Delegate Policy Management*

When you delegate access management for a realm, you might want to delegate policy management. You can delegate policy management by creating a referral.

1. In the OpenAM console, browse to Access Control > *Realm Name* > Policies, where *Realm Name* is the realm *from which* you intend to delegate policy.
2. Click the New Referral... button in the Policies table.
3. In the New Referral screen, provide at minimum a name for the referral.
4. Set up Rules to identify the resources that the Referral applies to.

When defining Rules to match resources, specify Resource Names as you would for Policies.

For example, you specify a URL Policy Agent rule to create policies for all resources under <http://example.com/> by creating rules for <http://example.com/>, for [http://example.com/\\*](http://example.com/), and also for [http://example.com/\\*?\\*](http://example.com/). When used at the end of a rule \* matches one or more characters, rather than zero or more characters.

5. Set up referrals to identify the realms *to which* to delegate policy management.

You can delegate to peer realms or sub realms (child realms), but not to parent realms.

6. Save your work.

At this point you can let the realm administrator know that she can create policies in her realm for the resources you specified in the rules of the referral.

#### Procedure 5.4. To Configure a Data Store for a Realm

When you first set up a realm, the new realm inherits the data store from the parent realm. Yet, if your administrators are in one realm and your users in another, your new child realm might retrieve users from a different data store.

1. In the OpenAM console, browse to Access Control > *Realm Name* > Data Stores.
2. Click New... in the Data Stores table to create a data store profile, and to provide the information needed to connect to the data store.
3. In the first screen, name the data store and select the type of data store to use.

Most data stores are directory services, though the Database Repository lets you connect to an SQL database through JDBC.

4. In the second screen, provide information on how to connect to your data store, and then click Finish to save your work.

See the [chapter on authentication](#) for hints on connecting to Active Directory, LDAP directory, and JDBC data sources.

5. Click the Subjects tab, and make sure the connection to your new data store is working, by searching for a known identity.

By default the Subjects list only retrieves 100 entries from the data store. Narrow your search if you do not see the identity you are looking for in the list.

6. If you no longer need the connection to the inherited data store *in this realm*, then you can delete its entry in the Data Stores table.

Also, once you change the data store for a realm, you might opt to change the [authentication module configuration](#) as described in the [chapter on authentication](#) to use your realm data store, rather than the inherited settings.

#### Procedure 5.5. To Configure a Web or J2EE Agent For Login to a Realm

You might choose to configure your agent in one realm, yet have your real users authenticate through another realm. In this case, you want your agents to redirect users to authenticate to their realm, rather than the agent realm.

1. In the OpenAM console, browse to Access Control > *Realm Name* > Agents > *Web or J2EE Agent Type* > *Agent Name* > OpenAM Services.
2. Add login and logout URLs, including the realm in the query string.

For example, if your *Realm Name* is `hr`, and you access OpenAM at `http://openam.example.com:8080/openam`:

- Login URL: `http://openam.example.com:8080/openam/UI/Login?realm=hr`
- Logout URL: `http://openam.example.com:8080/openam/UI/Logout?realm=hr`

3. Save your work.

## Chapter 6

# Configuring Policy Agent Profiles

You install policy agents in web servers and web application containers to enforce access policies OpenAM applies to protected web sites and web applications. Policy agents depend on OpenAM for all authentication and authorization decisions. Their primary responsibility consists in enforcing what OpenAM decides in a way that is unobtrusive to the user. In organizations with many servers, you might well install many policy agents.

Policy agents can have local configurations where they are installed, but usually you store all policy agent configuration information in the OpenAM configuration store, defining policy agent profiles for each, and then you let the policy agents access their profiles through OpenAM such that you manage all agent configuration changes centrally. This chapter describes how to set up policy agent profiles in OpenAM for centralized configuration.

## 6.1. Identity Gateway or Policy Agent?

OpenAM includes both the *Identity Gateway* and also a variety of policy agents. Both the *Identity Gateway* and also the policy agents enforce policy, redirecting users to authenticate when necessary, and controlling access to protected resources. Yet, the *Identity Gateway* runs as a self-contained reverse proxy located between the users and the protected applications. Policy agents are installed into the servers where applications run, intercepting requests in that context.

The *Identity Gateway* allows you to protect access to applications not suited for a policy agent. Not all web servers and Java EE applications have policy agents. Not all operating systems work with policy agents.

Policy agents have the advantage, where you can install them, of sitting within your existing server infrastructure. Once you have agents installed into the servers with web applications or sites to protect, then you can manage their configurations centrally from OpenAM.

Of course, for organizations with both servers where you can install policy agents and also applications that you must protect without touching the server, you can use policy agents on the former and the *Identity Gateway* for the latter.

## 6.2. Kinds of Agent Profiles

When you open the OpenAM console to configure agents for the top level realm, you can choose from a number of different types of agents. Web and J2EE policy agents are the most common, requiring the least integration effort.

### **Web**

You install web agents in web servers to protect web sites.

### **J2EE**

You install J2EE agents in web application containers to protect web applications.

### **Web Service Provider**

WSP agents are for use with Web Services Security.

### **Web Service Client**

WSC agents are for use with Web Services Security.

### **Discovery**

The Discovery Service agent has the trust authority configuration that OpenAM uses to communicate with a Liberty Discovery Service.

### **STS Client**

The Security Token Service client agent is for securing requests to the Security Token Service.

### **2.2 Agents**

Version 2.2 web and J2EE policy agents hold their configuration locally, connecting to OpenAM with a user name, password combination. This kind of agent is provided for backwards compatibility.

### **OAuth 2.0 Client Agent**

OAuth 2.0 clients are registered using this type of policy agent profile.

### **Agent Authenticator**

The agent authenticator can read agent profiles by connecting to OpenAM with a user name, password combination, but unlike the agent profile administrator, cannot change agent configuration.

## 6.3. Creating Agent Profiles

This section concerns creating agent profiles, and creating groups that let agents inherit settings when you have many agents with nearly the same profile settings.

### Procedure 6.1. To Create an Agent Profile

To create a new web or J2EE policy agent profile, you need a name and password for the agent, and the URLs to OpenAM and the application to protect.

1. On the Access Control tab page of the OpenAM console, click the link for the realm in which you manage agents.
2. Click the Agents tab, click the tab page for the kind of agent you want to create, and then click the New... button in the Agent table.
3. Provide a name for the agent, and also the URLs to OpenAM and to the application to protect, then click Create.

**New Agent** Create Cancel

\* Indicates required field

\* Name:

\* Password:

\* Re-Enter Password:

Configuration:  Local  Centralized  
Where agent properties are stored. Local is the server on which the agent is running. Centralized is the OpenAM Server

\* Server URL:   
protocol://host:port/deploymentUri e.g. http://opensso.sample.com:58080/opensso

\* Agent URL:   
protocol://host:port e.g. http://agent1.sample.com:1234

4. After creating the agent profile, you can click the link to the new profile to adjust and export the configuration.

### Procedure 6.2. To Create an Agent Profile Group & Inherit Settings

Agent profile groups let you set up multiple agents to inherit settings from the group. To create a new web or J2EE agent profile group, you need a name and the URL to the OpenAM server in which you store the profile.

1. On the Access Control tab page of the OpenAM console, click the link for the realm in which you manage agents.
2. Click the Agents tab, click the tab page for the kind of agent you want to create, and then click the New... button in the Group table.

After creating the group profile, you can click the link to the new group profile to fine-tune or export the configuration.

3. Inherit group settings by selecting your agent profile, and then selecting the group name in the Group drop-down list near the top of the profile page.

You can then adjust inheritance by clicking Inheritance Settings on the agent profile page.

### Procedure 6.3. To Create an Agent Profile Using the Command Line

You can create a policy agent profile in OpenAM using the **ssoadm** command-line tool. You do so by specifying the agent properties either as a list of attributes, or by using an agent properties file as shown below. Export an existing policy agent configuration before you start to see what properties you want to set when creating the agent profile.

The following procedure demonstrates creating a web policy agent profile using the **ssoadm** command.

1. Make sure the **ssoadm** command is installed as described in the *Installation Guide* procedure, *To Set Up Administration Tools* in the *Installation Guide*.
2. Determine the list of properties to set in the agent profile.

The following properties file shows a minimal configuration for a web policy agent profile.

```
$ cat myWebAgent.properties
com.sun.identity.agents.config.agenturi.prefix=http://www.example.com:80/amagent
com.sun.identity.agents.config.cdsso.cdcservlet.url[0]=https://openam.example.com:8443/openam/cdcservlet
com.sun.identity.agents.config.fqdn.default=www.example.com
com.sun.identity.agents.config.login.url[0]=http://openam.example.com:8443/openam/UI/Login
com.sun.identity.agents.config.logout.url[0]=http://openam.example.com:8443/openam/UI/Logout
com.sun.identity.agents.config.remote logfile=amAgent_www_example_com_80.log
com.sun.identity.agents.config.repository.location=centralized
com.sun.identity.client.notification.url=http://www.example.com:80/UpdateAgentCacheServlet?
shortcircuit=false
com.sun.identity.client.notification.url=http://www.example.com:80/UpdateAgentCacheServlet?
shortcircuit=false
sunIdentityServerDeviceKeyValue[0]=agentRootURL=http://www.example.com:80/
sunIdentityServerDeviceStatus=Active
userpassword=password
```

3. Set up a password file used when authenticating to OpenAM.

```
$ echo password > /tmp/pwd.txt
$ chmod 400 /tmp/pwd.txt
```

4. Create the profile in OpenAM.

```
$ ssoadm create-agent --realm /
--agentname myWebAgent --agenttype WebAgent --adminid amadmin
--password-file /tmp/pwd.txt --datafile myWebAgent.properties
```

Agent configuration was created.

At this point you can view the profile in OpenAM Console under Access Control > *Realm Name* > Agents to make sure the configuration is what you expect.

## 6.4. Delegating Agent Profile Creation

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

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

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

1. In OpenAM console, browse to Access Control > *Realm Name* > Subjects.
2. Under Group click New... and create a group for agent administrators.
3. Switch to the Privileges tab for the realm, and click the name of the group you created.
4. Select "Read and write access to all configured Agents," and then Save your work.
5. Return to the Subjects tab, and under User create as many agent administrator users as needed.
6. For each agent administrator user, edit the user profile.

Under the Group tab of the user profile, add the user to agent profile administrator group, and then Save your work.

7. Provide each system administrator who installs policy agents with their agent administrator credentials.

When installing the policy agent with the `--custom-install` option, the system administrator can choose the option to create the profile during installation, and then provide the agent administrator user name and the path to a read-only file containing the agent administrator password.

## 6.5. Configuring Web Policy Agents

When you create a web policy agent profile and install the agent, you can choose to store the agent configuration centrally and configure the agent through OpenAM console. Alternatively, you can choose to store the agent configuration locally and configure the agent by changing values in the

properties file. This section covers centralized configuration, indicating the corresponding properties for use in a local configuration file where applicable.<sup>1</sup>

**Tip**

To show the agent properties in configuration file format that correspond to what you see in the console, click Export Configuration after editing agent properties.

This corresponds to the local Java properties configuration file that is set up when you install an agent, for example in `Agent_001/config/OpenSSOAgentConfiguration.properties`.

After changing properties specified as "Hot swap: no" you must restart the agent for the changes to take effect.

### 6.5.1. Configuring Web Policy Agent Global Properties

This section covers global web agent properties. After creating the agent profile, you access these properties in the OpenAM console under Access Control > *Realm Name* > Agents > Web > *Agent Name* > Global.

#### *Profile properties*

**Group**

For assigning the agent to a previously configured web agent group in order to inherit selected properties from the group.

**Password**

Agent password used when creating the password file and when installing the agent.

**Status**

Status of the agent configuration.

**Location of Agent Configuration Repository**

Indicates agent's configuration located either on agent's host or centrally on OpenAM server.

If you change this to a local configuration, you can no longer manage the policy agent configuration through OpenAM console.

Property: `com.sun.identity.agents.config.repository.location`

**Agent Configuration Change Notification**

Enable agent to receive notification messages from OpenAM server for configuration changes.

<sup>1</sup>The configuration file syntax is that of a standard Java properties file, though backslash escapes can be used only to wrap long lines. See `java.util.Properties.load` for a description of the format. The value of a property specified multiple times is not defined.

Property: `com.sun.identity.agents.config.change.notification.enable`

### Enable Notifications

If enabled, the agent receives policy updates from the OpenAM notification mechanism to maintain its internal cache. If disabled, the agent must poll OpenAM for changes.

Property: `com.sun.identity.agents.config.notification.enable`

Hot swap: no

### Agent Notification URL

URL used by agent to register notification listeners.

Property: `com.sun.identity.client.notification.url`

Hot swap: no

### Agent Deployment URI Prefix

The default value is `agent-root-URL/amagent`.

Property: `com.sun.identity.agents.config.agenturi.prefix`

Hot swap: yes

### Configuration Reload Interval

Interval in minutes to fetch agent configuration from OpenAM. Used if notifications are disabled. Default: 60.

Property: `com.sun.identity.agents.config.polling.interval`

Hot swap: no

### Configuration Cleanup Interval

Interval in minutes to cleanup old agent configuration entries unless they are referenced by current requests. Default: 30.

Property: `com.sun.identity.agents.config.cleanup.interval`

Hot swap: no

### Agent Root URL for CDSSO

The agent root URL for CDSSO. The valid value is in the format `protocol://hostname:port/` where *protocol* represents the protocol used, such as `http` or `https`, *hostname* represents the host name

of the system where the agent resides, and *port* represents the port number on which the agent is installed. The slash following the port number is required.

If your agent system also has virtual host names, add URLs with the virtual host names to this list as well. OpenAM checks that `goto` URLs match one of the agent root URLs for CDSSO.

## General properties

### SSO Only Mode

When enabled, agent only enforces authentication (SSO), but no policies for authorization.

Property: `com.sun.identity.agents.config.sso.only`

### Resources Access Denied URL

The URL of the customized access denied page. If no value is specified (default), then the agent returns an HTTP status of 403 (Forbidden).

Property: `com.sun.identity.agents.config.access.denied.url`

### Agent Debug Level

Default is `Error`. Increase to `Message` or even `All` for fine-grained detail.

Property: `com.sun.identity.agents.config.debug.level`

You can set the level in the configuration file by module using the format `module[:level]` `[,module[:level:level]]*`, where *module* is one of `AuthService`, `NamingService`, `PolicyService`, `SessionService`, `PolicyEngine`, `ServiceEngine`, `Notification`, `PolicyAgent`, `RemoteLog`, or `all`, and *level* is one of the following.

- `0`: Disable logging from specified module  
At this level the agent nevertheless logs messages having the level value `always`.
- `1`: Log error messages
- `2`: Log warning and error messages
- `3`: Log info, warning, and error messages
- `4`: Log debug, info, warning, and error messages
- `5`: Like level 4, but with even more debugging messages

When you omit *level*, the agent uses the default level, which is the level associated with the `all` module.

The following example used in the local configuration sets the log overall level to debug for all messages.

```
com.sun.identity.agents.config.debug.level=all:4
```

## Agent Debug File Rotation

When enabled, rotate the debug file when specified file size is reached.

Property: `com.sun.identity.agents.config.debug.file.rotate`

## Agent Debug File Size

Debug file size in bytes beyond which the log file is rotated. The minimum is 1048576 bytes (1 MB), and lower values are reset to 1 MB. OpenAM console sets a default of 10 MB.

Property: `com.sun.identity.agents.config.debug.file.size`

## Audit properties

### Audit Access Types

Types of messages to log based on user URL access attempts.

Property: `com.sun.identity.agents.config.audit.accesstype`

Valid values for the configuration file property include `LOG_NONE`, `LOG_ALLOW`, `LOG_DENY`, and `LOG_BOTH`.

### Audit Log Location

Specifies where audit messages are logged. By default, audit messages are logged remotely.

Property: `com.sun.identity.agents.config.log.disposition`

Valid values for the configuration file property include `REMOTE`, `LOCAL`, and `ALL`.

### Remote Log Filename

Name of file stored on OpenAM server that contains agent audit messages if log location is remote or all.

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

Hot swap: no

### Remote Audit Log Interval

Periodic interval in minutes in which audit log messages are sent to the remote log file.

Property: `com.sun.identity.agents.config.remote.log.interval`

Default: 5

Hot swap: no

### Rotate Local Audit Log

When enabled, audit log files are rotated when reaching the specified size.

Property: `com.sun.identity.agents.config.local.log.rotate`

### Local Audit Log Rotation Size

Beyond this size limit in bytes the agent rotates the local audit log file if rotation is enabled.

Property: `com.sun.identity.agents.config.local.log.size`

Default: 50 MB

### *Fully Qualified Domain Name Checking properties*

#### FQDN Check

Enables checking of FQDN default value and FQDN map values.

Property: `com.sun.identity.agents.config.fqdn.check.enable`

#### FQDN Default

Fully qualified domain name that the users should use in order to access resources. Without this value, the web server can fail to start, thus you set the property on agent installation, and only change it when absolutely necessary.

This property ensures that when users access protected resources on the web server without specifying the FQDN, the agent can redirect the users to URLs containing the correct FQDN.

Property: `com.sun.identity.agents.config.fqdn.default`

#### FQDN Virtual Host Map

Enables virtual hosts, partial hostname and IP address to access protected resources. Maps invalid or virtual name keys to valid FQDN values so the agent can properly redirect users and the agents receive cookies belonging to the domain.

To map `myserver` to `myserver.mydomain.example`, enter `myserver` in the Map Key field, and enter `myserver.mydomain.example` in the Corresponding Map Value field. This corresponds to `com.sun.identity.agents.config.fqdn.mapping[myserver]=myserver.mydomain.example`.

Invalid FQDN values can cause the web server to become unusable or render resources inaccessible.

Property: `com.sun.identity.agents.config.fqdn.mapping`

## 6.5.2. Configuring Web Policy Agent Application Properties

This section covers application web agent properties. After creating the agent profile, you access these properties in the OpenAM console under Access Control > *Realm Name* > Agents > Web > *Agent Name* > Application.

### *Not Enforced URL Processing properties*

#### **Ignore Path Info for Not Enforced URLs**

When enabled, the path info and query are stripped from the request URL before being compared with the URLs of the not enforced list for those URLs containing a wildcard character. This prevents a user from accessing `http://host/index.html` by requesting `http://host/index.html/hack.gif` when the not enforced list includes `http://host/*.gif`.

#### **Note**

This setting is not supported by the Varnish Cache agent.

For a more generally applicable setting, see Ignore Path Info properties.

Property: `com.sun.identity.agents.config.ignore.path.info.for.not.enforced.list`

#### **Enable Regular Expressions for Not Enforced URLs (Not yet in OpenAM console)**

Enable use of Perl-compatible regular expressions in Not Enforced URL settings by using the following property under Advanced > Custom Properties in the agent profile.

```
com.forgerock.agents.notenforced.url.regex.enable=true
```

#### **Not Enforced URLs**

List of URLs for which no authentication is required. You can use wildcards to define a pattern for a URL.

The `*` wildcard matches all characters except question mark (`?`), cannot be escaped, and spans multiple levels in a URL. Multiple forward slashes do not match a single forward slash, so `*` matches `mult/iple/dirs`, yet `mult/*/dirs` does not match `mult/dirs`.

The `-*` wildcard matches all characters except forward slash (`/`) or question mark (`?`), and cannot be escaped. As it does not match `/`, `-*` does not span multiple levels in a URL.

OpenAM does not let you mix `*` and `-*` in the same URL.

Examples include `http://www.example.com/logout.html`, `http://www.example.com/images/*`, `http://www.example.com/css/*-`, and `http://www.example.com/*.jsp?locale=*`.

Trailing forward slashes are not recognized as part of a resource name. Therefore `http://www.example.com/images//` and `http://www.example.com/images` are equivalent.

Property: `com.sun.identity.agents.config.notenforced.url`

If you enabled use of Perl-compatible regular expressions to match Not Enforced URLs, then all your settings must be done using regular expressions. (Do not mix settings; use either the mechanism described above or Perl-compatible regular expressions, but not both.)

The following example shows settings where no authentication is required for URLs whose path ends `/publicA` or `/publicB` (with or without query string parameters), and no authentication is required to access `.png`, `.jpg`, `.gif`, `.js`, or `.css` files under URLs that do not contain `/protectedA/` or `/protectedB/`.

```
.*/(PublicServletA|PublicServletB)(\?.*|$)  
^(?!.*(protectedA|protectedB/)).*\.(png|jpg|gif|js|css)(\?.*|$)
```

### Invert Not Enforced URLs

Only enforce not enforced list of URLs. In other words, enforce policy only for those URLs and patterns specified in the list.

Property: `com.sun.identity.agents.config.notenforced.url.invert`

### Fetch Attributes for Not Enforced URLs

When enabled, the agent fetches profile, response, and session attributes that are mapped by doing policy evaluation, and forwards these attributes to not enforced URLs.

Property: `com.sun.identity.agents.config.notenforced.url.attributes.enable`

### *Not Enforced IP Processing properties*

#### Not Enforced Client IP List

No authentication and authorization are required for the requests coming from these client IP addresses.

Property: `com.sun.identity.agents.config.notenforced.ip`

#### CIDR Client IP Specification (Not yet in OpenAM console)

As of version 3.0.4, web policy agents with this property set to `cidr` can use IPv4 netmasks and IP ranges instead of wildcards as values for Not Enforced Client IP addresses. Version 3.0.5 adds support for IPv6, including the IPv6 loopback address, `::1`.

When the parameter is defined, wildcards are ignored in Not Enforced Client IP settings. Instead, you can use settings such as those shown in the following examples.

#### Netmask Example

To disable policy agent enforcement for addresses in 192.168.1.1 to 192.168.1.255, use the following setting.

```
com.sun.identity.agents.config.notenforced.ip = 192.168.1.1/24
```

The following example shows an IPv6 address with a corresponding network mask.

```
com.sun.identity.agents.config.notenforced.ip = 2001:5c0:9168:0:0:0:2/128
```

Currently the policy agent stops evaluating properties after reaching an invalid netmask in the list.

### IP Range Example

To disable policy agent enforcement for addresses between 192.168.1.1 to 192.168.4.3 inclusive, use the following setting.

```
com.sun.identity.agents.config.notenforced.ip = 192.168.1.1-192.168.4.3
```

The following example shows a range of IPv6 addresses.

```
com.sun.identity.agents.config.notenforced.ip = 2001:5c0:9168:0:0:0:0:1-2001:5c0:9168:0:0:0:0:2
```

Property: `com.forgerock.agents.config.notenforced.ip.handler`

Hot swap: no

### Client IP Validation

When enabled, validate that the subsequent browser requests come from the same IP address that the SSO token is initially issued against.

Property: `com.sun.identity.agents.config.client.ip.validation.enable`

## Profile Attributes Processing properties

### Profile Attribute Fetch Mode

When set to `HTTP_COOKIE` or `HTTP_HEADER`, profile attributes are introduced into the cookie or the headers, respectively.

Property: `com.sun.identity.agents.config.profile.attribute.fetch.mode`

### Profile Attribute Map

Maps the profile attributes to HTTP headers for the currently authenticated user. Map Keys are LDAP attribute names, and Map Values are HTTP header names.

To populate the value of profile attribute CN under `CUSTOM-Common-Name`: enter CN in the Map Key field, and enter `CUSTOM-Common-Name` in the Corresponding Map Value field. This corresponds to `com.sun.identity.agents.config.profile.attribute.mapping[cn]=CUSTOM-Common-Name`.

In most cases, in a destination application where an HTTP header name shows up as a request header, it is prefixed by `HTTP_`, lower case letters become upper case, and hyphens (-) become underscores (\_). For example, `common-name` becomes `HTTP_COMMON_NAME`.

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

## *Response Attributes Processing properties*

### **Response Attribute Fetch Mode**

When set to `HTTP_COOKIE` or `HTTP_HEADER`, response attributes are introduced into the cookie or the headers, respectively.

Property: `com.sun.identity.agents.config.response.attribute.fetch.mode`

### **Response Attribute Map**

Maps the policy response attributes to HTTP headers for the currently authenticated user. The response attribute is the attribute in the policy response to be fetched.

To populate the value of response attribute `uid` under `CUSTOM-User-Name`: enter `uid` in the Map Key field, and enter `CUSTOM-User-Name` in the Corresponding Map Value field. This corresponds to `com.sun.identity.agents.config.response.attribute.mapping[uid]=Custom-User-Name`.

In most cases, in a destination application where an HTTP header name shows up as a request header, it is prefixed by `HTTP_`, lower case letters become upper case, and hyphens (-) become underscores (\_). For example, `response-attr-one` becomes `HTTP_RESPONSE_ATTR_ONE`.

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

## *Session Attributes Processing properties*

### **Session Attribute Fetch Mode**

When set to `HTTP_COOKIE` or `HTTP_HEADER`, session attributes are introduced into the cookie or the headers, respectively.

Property: `com.sun.identity.agents.config.session.attribute.fetch.mode`

### **Session Attribute Map**

Maps session attributes to HTTP headers for the currently authenticated user. The session attribute is the attribute in the session to be fetched.

To populate the value of session attribute `UserToken` under `CUSTOM-userid`: enter `UserToken` in the Map Key field, and enter `CUSTOM-userid` in the Corresponding Map Value field. This corresponds to `com.sun.identity.agents.config.session.attribute.mapping[UserToken]=CUSTOM-userid`.

In most cases, in a destination application where an HTTP header name shows up as a request header, it is prefixed by `HTTP_`, lower case letters become upper case, and hyphens (-) become underscores (\_). For example, `success-url` becomes `HTTP_SUCCESS_URL`.

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

## Common Attributes Fetching Processing properties

### Attribute Multi Value Separator

Specifies separator for multiple values. Applies to all types of attributes such as profile, session and response attributes. Default: |.

Property: `com.sun.identity.agents.config.attribute.multi.value.separator`

## 6.5.3. Configuring Web Policy Agent SSO Properties

This section covers SSO web agent properties. After creating the agent profile, you access these properties in the OpenAM console under Access Control > *Realm Name* > Agents > Web > *Agent Name* > SSO

### Cookie properties

#### Cookie Name

Name of the SSO Token cookie used between the OpenAM server and the agent. Default: `iPlanetDirectoryPro`.

Property: `com.sun.identity.agents.config.cookie.name`

Hot swap: no

#### Cookie Security

When enabled, the agent marks cookies secure, sending them only if the communication channel is secure.

Property: `com.sun.identity.agents.config.cookie.secure`

Hot swap: no

#### HTTPOnly Cookies (Not yet in OpenAM console)

As of version 3.0.5, web policy agents with this property set to `true` mark cookies as HTTPOnly, to prevent scripts and third-party programs from accessing the cookies.

Property: `com.sun.identity.cookie.httponly`

### Cross Domain SSO properties

#### Cross Domain SSO

Enables Cross Domain Single Sign On.

Property: `com.sun.identity.agents.config.cdsso.enable`

## CDSSO Servlet URL

List of URLs of the available CDSSO controllers that the agent can use for CDSSO processing. For example, `http://openam.example.com:8080/openam/cdcervlet`.

Property: `com.sun.identity.agents.config.cdsso.cdcervlet.url`

## Cookies Domain List

List of domains, such as `.example.com`, in which cookies have to be set in CDSSO. If this property is left blank, then the fully qualified domain name of the cookie for the agent server is used to set the cookie domain, meaning that a host cookie rather than a domain cookie is set.

To set the list to `.example.com`, and `.example.net` using the configuration file property, include the following.

```
com.sun.identity.agents.config.cdsso.cookie.domain[0]=.example.com
com.sun.identity.agents.config.cdsso.cookie.domain[1]=.example.net
```

Property: `com.sun.identity.agents.config.cdsso.cookie.domain`

## Cookie Reset properties

### Cookie Reset

When enabled, agent resets cookies in the response before redirecting to authentication.

Property: `com.sun.identity.agents.config.cookie.reset.enable`

### Cookie Reset Name List

List of cookies in the format `name[=value][;Domain=value]`.

Concrete examples include the following with two list items configured.

- `LtpaToken`, corresponding to `com.sun.identity.agents.config.cookie.reset[0]=LtpaToken`. The default domain is taken from FQDN Default.
- `token=value;Domain=subdomain.domain.com`, corresponding to `com.sun.identity.agents.config.cookie.reset[1]= token=value;Domain=subdomain.domain.com`

Property: `com.sun.identity.agents.config.cookie.reset`

## 6.5.4. Configuring Web Policy Agent OpenAM Services Properties

This section covers OpenAM services web agent properties. After creating the agent profile, you access these properties in the OpenAM console under Access Control > *Realm Name* > Agents > Web > *Agent Name* > OpenAM Services.

## Login URL properties

### OpenAM Login URL

OpenAM login page URL, such as `http://openam.example.com:8080/openam/UI/Login`, to which the agent redirects incoming users without sufficient credentials so then can authenticate.

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

### OpenAM Conditional Login URL (Not yet in OpenAM console)

To conditionally redirect users based on the incoming request URL, set this property.

This takes the incoming request domain to match, a vertical bar ( | ), and then a comma-separated list of URLs to which to redirect incoming users.

If the domain before the vertical bar matches an incoming request URL, then the policy agent uses the list of URLs to determine how to redirect the user-agent. If the global property FQDN Check (`com.sun.identity.agents.config.fqdn.check.enable`) is enabled for the policy agent, then the policy agent iterates through the list until it finds an appropriate redirect URL that matches the FQDN check. Otherwise, the policy agent redirects the user-agent to the first URL in the list.

Property: `com.forgerock.agents.conditional.login.url`

Examples: `com.forgerock.agents.conditional.login.url[0]= login.example.com|http://openam1.example.com/openam/UI/Login, http://openam2.example.com/openam/UI/Login, com.forgerock.agents.conditional.login.url[1]= signin.example.com|http://openam3.example.com/openam/UI/Login, http://openam4.example.com/openam/UI/Login`

If CDSSO is enabled for the policy agent, then this property takes CDSSO Servlet URLs for its values (`com.sun.identity.agents.config.cdssso.cdcservlet.url`), rather than OpenAM login URLs.

CDSSO examples: `com.forgerock.agents.conditional.login.url[0]= login.example.com|http://openam1.example.com/openam/cdcservlet, http://openam2.example.com/openam/cdcservlet, com.forgerock.agents.conditional.login.url[1]= signin.example.com|http://openam3.example.com/openam/cdcservlet, http://openam4.example.com/openam/cdcservlet`

### Agent Connection Timeout

Timeout period in seconds for an agent connection with OpenAM auth server.

Property: `com.sun.identity.agents.config.auth.connection.timeout`

Default: 2

### Polling Period for Primary Server

Interval in minutes, agent polls to check the primary server is up and running. Default: 5.

Property: `com.sun.identity.agents.config.poll.primary.server`

Hot swap: no

## Logout URL properties

### OpenAM Logout URL

OpenAM logout page URL, such as `http://openam.example.com:8080/openam/UI/Logout`.

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

### Enable Logout URL Redirect (Not yet in OpenAM console)

Logout URL redirect is enabled by default.

When this is disabled, instead of redirecting the user-agent, the policy agent performs session logout in the background and then continues processing access to the current URL. Disable this using Advanced > Custom Properties in the agent profile.

```
com.forgerock.agents.config.logout.redirect.disable=true
```

## Agent Logout URL properties

### Logout URL List

List of application logout URLs, such as `http://www.example.com/logout.html`. The user is logged out of the OpenAM session when these URLs are accessed. When using this property, specify a value for the Logout Redirect URL property.

Property: `com.sun.identity.agents.config.agent.logout.url`

### Agent Logout URL Regular Expression (Not yet in OpenAM console)

Perl-compatible regular expression that matches logout URLs. Set this using Advanced > Custom Properties in the agent profile.

For example, to match URLs with `protectedA` or `protectedB` in the path and `op=logout` in the query string, use the following setting.

```
com.forgerock.agents.agent.logout.url.regex= \
.*(/protectedA\?|/protectedB\?/).*{\&op=logout\&}(.*$)
```

When you use this property, the agent ignores the settings for Logout URL List.

### Logout Cookies List for Reset

Cookies to be reset upon logout in the same format as the cookie reset list.

Property: `com.sun.identity.agents.config.logout.cookie.reset`

## Logout Redirect URL

User gets redirected to this URL after logout. Specify this property alongside a Logout URL List.

Property: `com.sun.identity.agents.config.logout.redirect.url`

## *Policy Client Service properties*

### Policy Cache Polling Period

Polling interval in minutes during which an entry remains valid after being added to the agent's cache.

Property: `com.sun.identity.agents.config.policy.cache.polling.interval`

Hot swap: no

### SSO Cache Polling Period

Polling interval in minutes during which an SSO entry remains valid after being added to the agent's cache.

Property: `com.sun.identity.agents.config.sso.cache.polling.interval`

Hot swap: no

### User ID Parameter

Agent sets this value for User Id passed in the session from OpenAM to the REMOTE\_USER server variable. Default: UserToken.

Property: `com.sun.identity.agents.config.userid.param`

### User ID Parameter Type

User ID can be fetched from either SESSION and LDAP attributes. Default: `SESSION`.

Property: `com.sun.identity.agents.config.userid.param.type`

### Fetch Policies from Root Resource

When enabled, the agent caches the policy decision of the resource and all resources from the root of the resource down. For example, if the resource is `http://host/a/b/c`, then the root of the resource is `http://host/`. This setting can be useful when a client is expect to access multiple resources on the same path. Yet, caching can be expensive if very many policies are defined for the root resource.

Property: `com.sun.identity.agents.config.fetch.from.root.resource`

Default: false

Hot swap: no

### Retrieve Client Hostname

When enabled, get the client hostname through DNS reverse lookup for use in policy evaluation. This setting can impact performance.

Property: `com.sun.identity.agents.config.get.client.host.name`

### Policy Clock Skew

Time in seconds used adjust time difference between agent system and OpenAM. Clock skew in seconds = AgentTime - OpenAMServerTime.

Use this property to adjust for small time differences encountered despite use of a time synchronization service. When this property is not set and agent time is greater than OpenAM server time, the agent can make policy calls to the OpenAM server before the policy subject cache has expired, or you can see infinite redirection occur.

Property: `com.sun.identity.agents.config.policy.clock.skew`

Hot swap: no

## 6.5.5. Configuring Web Policy Agent Miscellaneous Properties

This section covers miscellaneous web agent properties. After creating the agent profile, you access these properties in the OpenAM console under Access Control > *Realm Name* > Agents > Web > *Agent Name* > Miscellaneous.

### *Advice Handling properties*

#### **Composite Advice Handling (Not yet in OpenAM console)**

As of version 3.0.4, when set to `true`, the agent sends composite advice in the query (GET request) instead of sending it through a POST request.

Property: `com.sun.am.use_redirect_for_advice`

### *Locale properties*

#### **Agent Locale**

The default locale for the agent.

Property: `com.sun.identity.agents.config.locale`

Hot swap: no

## *Anonymous user properties*

### **Anonymous User**

Enable or disable REMOTE\_USER processing for anonymous users.

Property: `com.sun.identity.agents.config.anonymous.user.enable`

## *Cookie Processing properties*

### **Encode special chars in Cookies**

When enabled, encode special chars in cookie by URL encoding. This is useful when profile, session, and response attributes contain special characters, and the attributes fetch mode is set to `HTTP_COOKIE`.

Property: `com.sun.identity.agents.config.encode.cookie.special.chars.enable`

### **Profile Attributes Cookie Prefix**

Sets cookie prefix in the attributes headers. Default: `HTTP_`.

Property: `com.sun.identity.agents.config.profile.attribute.cookie.prefix`

### **Profile Attributes Cookie Maxage**

Maximum age in seconds of custom cookie headers. Default: 300.

Property: `com.sun.identity.agents.config.profile.attribute.cookie.maxage`

## *URL Handling properties*

### **URL Comparison Case Sensitivity Check**

When enabled, enforces case insensitivity in both policy and not enforced URL evaluation.

Property: `com.sun.identity.agents.config.url.comparison.case.ignore`

### **Encode URL's Special Characters**

When enabled, encodes the URL which has special characters before doing policy evaluation.

Property: `com.sun.identity.agents.config.encode.url.special.chars.enable`

## *Ignore Naming URL properties*

### **Ignore Preferred Naming URL in Naming Request**

When enabled, do not send a preferred naming URL in the naming request.

Property: `com.sun.identity.agents.config.ignore.preferred.naming.url`

## *Invalid URL properties (Not yet in OpenAM console)*

### **Invalid URL Regular Expression**

Use a Perl-compatible regular expression to filter out invalid request URLs. The policy agent rejects requests to invalid URLs with HTTP 403 Forbidden status without further processing. Use **Advanced > Custom Properties** to set this in the agent profile.

For example, to filter out URLs containing the symbols in the list `./, /., /, ., \, %00-%1f, %7f-%ff, %25, %2B, %2C, %7E, .info`, use the following setting.

```
com.forgerock.agents.agent.invalid.url.regex= \
^(?!(|/\.\./|*|\.\info|%25|%2B|%2C|[%0-1][0-9a-fA-F]|%[7-9a-fA-F][0-9a-fA-F])).)$
```

## *Ignore Server Check properties*

### **Ignore Server Check**

When enabled, do not check whether OpenAM is up before doing a 302 redirect.

Property: `com.sun.identity.agents.config.ignore.server.check`

## *Ignore Path Info properties*

### **Ignore Path Info in Request URL**

When enabled, strip path info from the request URL while doing the Not Enforced List check, and URL policy evaluation. This is designed to prevent a user from accessing a URI by appending the matching pattern in the policy or not enforced list.

#### **Note**

This setting is not supported by the Varnish Cache agent.

For example, if the not enforced list includes `http://host/*.gif`, then stripping path info from the request URI prevents access to `http://host/index.html` by using `http://host/index.html?hack.gif`.

However, when a web server is configured as a reverse proxy for a J2EE application server, the path info is interpreted to map a resource on the proxy server rather than the application server. This prevents the not enforced list or the policy from being applied to the part of the URI below the application server path if a wildcard character is used.

For example, if the not enforced list includes `http://host/webapp/servcontext/*` and the request URL is `http://host/webapp/servcontext/example.jsp`, the path info is `/servcontext/example.jsp` and the resulting request URL with path info stripped is `http://host/webapp/`, which does not match the not

enforced list. Thus when this property is enabled, path info is not stripped from the request URL even if there is a wildcard in the not enforced list or policy.

Make sure therefore when this property is enabled that there is nothing following the wildcard in the not enforced list or policy.

Property: `com.sun.identity.agents.config.ignore.path.info`

### *Multi-byte Enable properties*

#### **Native Encoding of Profile Attributes**

When enabled, the agent encodes the LDAP header values in the default encoding of operating system locale. When disabled, the agent uses UTF-8.

Property: `com.sun.identity.agents.config.convert.mbyte.enable`

### *Goto Parameter Name properties*

#### **Goto Parameter Name**

Property used only when CDSSO is enabled. Only change the default value, `goto` when the login URL has a landing page specified such as, `com.sun.identity.agents.config.cdsso.cdcervlet.url = http://openam.example.com:8080/openam/cdcervlet?goto= http://www.example.com/landing.jsp`. The agent uses this parameter to append the original request URL to this cdcervlet URL. The landing page consumes this parameter to redirect to the original URL.

As an example, if you set this value to `goto2`, then the complete URL sent for authentication is `http://openam.example.com:8080/openam/cdcervlet?goto= http://www.example.com/landing.jsp?goto2=http://www.example.com/original.jsp`.

Property: `com.sun.identity.agents.config.redirect.param`

### *Deprecated Agent properties*

#### **Anonymous User Default Value**

User ID of unauthenticated users. Default: `anonymous`.

Property: `com.sun.identity.agents.config.anonymous.user.id`

## 6.5.6. Configuring Web Policy Agent Advanced Properties

This section covers advanced web agent properties. After creating the agent profile, you access these properties in the OpenAM console under Access Control > *Realm Name* > Agents > Web > *Agent Name* > Advanced.

## *Client Identification properties*

If the agent is behind a proxy or load balancer, then the agent can get client IP and host name values from the proxy or load balancer. For proxies and load balancer that support providing the client IP and host name in HTTP headers, you can use the following properties.

When multiple proxies or load balancers sit in the request path, the header values can include a comma-separated list of values with the first value representing the client, as in `client,next-proxy,first-proxy`.

### **Client IP Address Header**

HTTP header name that holds the IP address of the client.

Property: `com.sun.identity.agents.config.client.ip.header`

### **Client Hostname Header**

HTTP header name that holds the hostname of the client.

Property: `com.sun.identity.agents.config.client.hostname.header`

## *Load Balancer properties*

### **Load Balancer Setup**

Enable if a load balancer is used for OpenAM services.

Property: `com.sun.identity.agents.config.load.balancer.enable`

Hot swap: no

### **Override Request URL Protocol**

Enable if the agent is sitting behind a SSL/TLS off-loader, load balancer, or proxy such that the protocol users use is different from the protocol the agent uses. When enabled, the protocol is overridden with the value from the Agent Deployment URI Prefix (property: `com.sun.identity.agents.config.agenturi.prefix`).

Property: `com.sun.identity.agents.config.override.protocol`

### **Override Request URL Host**

Enable if the agent is sitting behind a SSL/TLS off-loader, load balancer, or proxy such that the host name users use is different from the host name the agent uses. When enabled, the host is overridden with the value from the Agent Deployment URI Prefix (property: `com.sun.identity.agents.config.agenturi.prefix`).

Property: `com.sun.identity.agents.config.override.host`

### Override Request URL Port

Enable if the agent is sitting behind a SSL/TLS off-loader, load balancer, or proxy such that the port users use is different from the port the agent uses. When enabled, the port is overridden with the value from the Agent Deployment URI Prefix (property: `com.sun.identity.agents.config.agenturi.prefix`).

Property: `com.sun.identity.agents.config.override.port`

### Override Notification URL

Enable if the agent is sitting behind a SSL/TLS off-loader, load balancer, or proxy such that the URL users use is different from the URL the agent uses. When enabled, the URL is overridden with the value from the Agent Deployment URI Prefix (property: `com.sun.identity.agents.config.agenturi.prefix`).

Property: `com.sun.identity.agents.config.override.notification.url`

## Post Data Preservation properties

### POST Data Preservation

Enables HTTP POST data preservation. This feature is available in the Apache 2.2, Microsoft IIS 6, Microsoft IIS 7, and Sun Java System Web Server web policy agents as of version 3.0.3.

Property: `com.sun.identity.agents.config.postdata.preserve.enable`

### POST Data Entries Cache Period

POST cache entry lifetime in minutes. Default: 10.

Property: `com.sun.identity.agents.config.postcache.entry.lifetime`

### POST Data Preservation Cookie Name (Not yet in OpenAM Console)

When HTTP POST data preservation is enabled, override properties are set to true, and the agent is behind a load balancer, then this property sets the name and value of the sticky cookie to use.

Property: `com.sun.identity.agents.config.postdata.preserve.lbcookie`

### Post Data Preservation URI Prefix (Not yet in OpenAM Console)

If you run multiple web servers with policy agents behind a load balancer that directs traffic based on the request URI, and you need to preserve POST data, then set this property.

By default, policy agents use a dummy URL for POST data preservation, `http://agent.host:port/dummypost/sunpostpreserve`, to handle POST data across redirects to and from OpenAM. When you set this property, the policy agent prefixes the property value to the dummy URL path. In other

words, when you set `com.forgerock.agents.config.pdpuri.prefix = appl`, the policy agent uses the dummy URL, `http://agent.host:port/appl/dummypost/sunpostpreserve`.

Next, use the prefix you set when you define load balancer URI rules. This ensures that clients end up being redirected to the policy agent that preserved the POST data.

Property: `com.forgerock.agents.config.pdpuri.prefix`

## *Sun Java System Proxy Server properties*

### **Override Proxy Server's Host and Port**

When enabled ignore the host and port settings.

Property: `com.sun.identity.agents.config.proxy.override.host.port`

Hot swap: no

## *Microsoft IIS Server properties*

### **Authentication Type**

The agent should normally perform authentication, so this is not required. If necessary, set to `none`.

Property: `com.sun.identity.agents.config.iis.auth.type`

Hot swap: no

### **Replay Password Key**

DES key for decrypting the basic authentication password in the session.

Property: `com.sun.identity.agents.config.replaypasswd.key`

### **Filter Priority**

The loading priority of filter, DEFAULT, HIGH, LOW, or MEDIUM.

Property: `com.sun.identity.agents.config.iis.filter.priority`

### **Filter configured with OWA**

Enable if the IIS agent filter is configured for OWA.

Property: `com.sun.identity.agents.config.iis.owa.enable`

### **Change URL Protocol to https**

Enable to avoid IE6 security pop-ups.

Property: `com.sun.identity.agents.config.iis.owa.enable.change.protocol`

### Idle Session Timeout Page URL

URL of the local idle session timeout page.

Property: `com.sun.identity.agents.config.iis.owa.enable.session.timeout.url`

## IBM Lotus Domino Server properties

### Check User in Domino Database

When enabled, the agent checks whether the user exists in the Domino name database.

Property: `com.sun.identity.agents.config.domino.check.name.database`

### Use LTPA token

Enable if the agent needs to use LTPA Token.

Property: `com.sun.identity.agents.config.domino.ltpa.enable`

### LTPA Token Cookie Name

The name of the cookie that contains the LTPA token.

Property: `com.sun.identity.agents.config.domino.ltpa.cookie.name`

### LTPA Token Configuration Name

The configuration name that the agent uses in order to employ the LTPA token mechanism.

Property: `com.sun.identity.agents.config.domino.ltpa.config.name`

### LTPA Token Organization Name

The organization name to which the LTPA token belongs.

Property: `com.sun.identity.agents.config.domino.ltpa.org.name`

## Custom properties

### Custom Properties

Additional properties to augment the set of properties supported by agent. Such properties take the following forms.

- `customproperty=custom-value1`

- `customlist[0]=customlist-value-0`
- `customlist[1]=customlist-value-1`
- `custommap[key1]=custommap-value-1`
- `custommap[key2]=custommap-value-2`

Property: `com.sun.identity.agents.config.freeformproperties`

## 6.6. Configuring J2EE Policy Agents

When you create a J2EE policy agent profile and install the agent, you can choose to store the agent configuration centrally and configure the agent through OpenAM console. Alternatively, you can choose to store the agent configuration locally and configure the agent by changing values in the properties file. This section covers centralized configuration, indicating the corresponding properties for use in a local configuration file where applicable.<sup>2</sup>

### Tip

To show the agent properties in configuration file format that correspond to what you see in the console, click Export Configuration after editing agent properties.

After changing properties specified as "Hot swap: no" you must restart the agent.

### 6.6.1. Configuring J2EE Policy Agent Global Properties

This section covers global web agent properties. After creating the agent profile, you access these properties in the OpenAM console under Access Control > *Realm Name* > Agents > J2EE > *Agent Name* > Global.

#### *Profile properties*

#### **Group**

For assigning the agent to a previously configured web agent group in order to inherit selected properties from the group.

#### **Password**

Agent password used when creating the password file and when installing the agent.

<sup>2</sup>The configuration file syntax is that of a standard Java properties file, though backslash escapes can be used only to wrap long lines. See `java.util.Properties.load` for a description of the format. The value of a property specified multiple times is not defined.

## Status

Status of the agent configuration.

## Agent Notification URL

URL used by agent to register notification listeners.

Property: `com.sun.identity.client.notification.url`

Hot swap: no

## Location of Agent Configuration Repository

Indicates agent's configuration located either on agent's host or centrally on OpenAM server.

If you change this to a local configuration, you can no longer manage the policy agent configuration through OpenAM console.

Property: `com.sun.identity.agents.config.repository.location`

## Configuration Reload Interval

Interval in seconds to fetch agent configuration from OpenAM. Used if notifications are disabled.  
Default: 0

Property: `com.sun.identity.agents.config.load.interval`

## Agent Configuration Change Notification

Enable agent to receive notification messages from OpenAM server for configuration changes.

Property: `com.sun.identity.agents.config.change.notification.enable`

## Agent Root URL for CDSSO

The agent root URL for CDSSO. The valid value is in the format `protocol://hostname:port/` where *protocol* represents the protocol used, such as `http` or `https`, *hostname* represents the host name of the system where the agent resides, and *port* represents the port number on which the agent is installed. The slash following the port number is required.

If your agent system also has virtual host names, add URLs with the virtual host names to this list as well. OpenAM checks that `goto` URLs match one of the agent root URLs for CDSSO.

## General properties

### Agent Filter Mode

Specifies how the agent filters requests to protected web applications. The global value functions as a default, and applies for protected applications that do not have their own filter settings. Valid settings include the following.

**ALL**

Enforce both the J2EE policy defined for the web container where the protected application runs, and also OpenAM policies.

When setting the filter mode to **ALL**, set the Map Key, but do not set any Corresponding Map Value.

**J2EE\_POLICY**

Enforce only the J2EE policy defined for the web container where the protected application runs.

**NONE**

Do not enforce policies to protect resources. In other words, turn off access management. Not for use in production.

**SSO\_ONLY**

Enforce only authentication, not policies.

**URL\_POLICY**

Enforce only OpenAM, URL resource based policies.

When setting the filter mode to **URL\_POLICY**, set the Map Key to the application name and the Corresponding Map Value to **URL\_POLICY**.

Property: `com.sun.identity.agents.config.filter.mode`

Hot swap: no

**HTTP Session Binding**

When enabled the agent invalidates the HTTP session upon login failure, when the user has no SSO session, or when the principal user name does not match the SSO user name.

Property: `com.sun.identity.agents.config.httpsession.binding`

**Login Attempt Limit**

When set to a value other than zero, this defines the maximum number of failed login attempts allowed during a single browser session, after which the agent blocks requests from the user.

Property: `com.sun.identity.agents.config.login.attempt.limit`

**Custom Response Header**

Specifies the custom headers the agent sets for the client. The key is the header name. The value is the header value.

Property: `com.sun.identity.agents.config.response.header`

For example, `com.sun.identity.agents.config.response.header[Cache-Control]=no-cache`.

### Redirect Attempt Limit

When set to a value other than zero, this defines the maximum number of redirects allowed for a single browser session, after which the agent blocks the request.

Property: `com.sun.identity.agents.config.redirect.attempt.limit`

### Agent Debug Level

Default is `Error`. Increase to `Message` for fine-grained detail.

Property: `com.iplanet.services.debug.level`

### *User Mapping properties*

#### User Mapping Mode

Specifies the mechanism used to determine the user ID.

Property: `com.sun.identity.agents.config.user.mapping.mode`

#### User Attribute Name

Specifies the data store attribute that contains the user ID.

Property: `com.sun.identity.agents.config.user.attribute.name`

#### User Principal Flag

When enabled, OpenAM uses both the principal user name and also the user ID for authentication.

Property: `com.sun.identity.agents.config.user.principal`

#### User Token Name

Specifies the session property name for the authenticated user's ID. Default: `UserToken`.

Property: `com.sun.identity.agents.config.user.token`

### *Audit properties*

#### Audit Access Types

Types of messages to log based on user URL access attempts.

Property: `com.sun.identity.agents.config.audit.accesstype`

Valid values for the configuration file property include `LOG_NONE`, `LOG_ALLOW`, `LOG_DENY`, and `LOG_BOTH`.

### Audit Log Location

Specifies where audit messages are logged. By default, audit messages are logged remotely.

Property: `com.sun.identity.agents.config.log.disposition`

Valid values for the configuration file property include `REMOTE`, `LOCAL`, and `ALL`.

### Remote Log Filename

Name of file stored on OpenAM server that contains agent audit messages if log location is remote or all.

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

Hot swap: no

### Rotate Local Audit Log

When enabled, audit log files are rotated when reaching the specified size.

Property: `com.sun.identity.agents.config.local.log.rotate`

### Local Audit Log Rotation Size

Beyond this size limit in bytes the agent rotates the local audit log file if rotation is enabled.

Property: `com.sun.identity.agents.config.local.log.size`

Default: 50 MB

## Fully Qualified Domain Name Checking properties

### FQDN Check

Enables checking of FQDN default value and FQDN map values.

Property: `com.sun.identity.agents.config.fqdn.check.enable`

### FQDN Default

Fully qualified domain name that the users should use in order to access resources.

This property ensures that when users access protected resources on the web server without specifying the FQDN, the agent can redirect the users to URLs containing the correct FQDN.

Property: `com.sun.identity.agents.config.fqdn.default`

## FQDN Virtual Host Map

Enables virtual hosts, partial hostname and IP address to access protected resources. Maps invalid or virtual name keys to valid FQDN values so the agent can properly redirect users and the agents receive cookies belonging to the domain.

To map `myserver` to `myserver.mydomain.example`, enter `myserver` in the Map Key field, and enter `myserver.mydomain.example` in the Corresponding Map Value field. This corresponds to `com.sun.identity.agents.config.fqdn.mapping[myserver]=myserver.mydomain.example`.

Property: `com.sun.identity.agents.config.fqdn.mapping`

## 6.6.2. Configuring J2EE Policy Agent Application Properties

This section covers application web agent properties. After creating the agent profile, you access these properties in the OpenAM console under Access Control > *Realm Name* > Agents > J2EE > *Agent Name* > Application.

### *Login Processing properties*

#### **Login Form URI**

Specifies the list of absolute URIs corresponding to a protected application's `web.xml form-login-page` element, such as `/myApp/jsp/login.jsp`.

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

#### **Login Error URI**

Specifies the list of absolute URIs corresponding to a protected application's `web.xml form-error-page` element, such as `/myApp/jsp/error.jsp`.

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

#### **Use Internal Login**

When enabled, the agent uses the internal default content file for the login.

Property: `com.sun.identity.agents.config.login.use.internal`

#### **Login Content File Name**

Full path name to the file containing custom login content when Use Internal Login is enabled.

Property: `com.sun.identity.agents.config.login.content.file`

## Logout Processing properties

### Application Logout Handler

Specifies how logout handlers map to specific applications. The key is the web application name. The value is the logout handler class.

To set a global logout handler for applications without other logout handlers defined, leave the key empty and set the value to the global logout handler class name, `GlobalApplicationLogoutHandler`.

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

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

### Application Logout URI

Specifies request URIs that indicate logout events. The key is the web application name. The value is the application logout URI.

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

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

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

### Logout Request Parameter

Specifies parameters in the HTTP request that indicate logout events. The key is the web application name. The value is the logout request parameter.

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

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

Property: `com.sun.identity.agents.config.logout.request.param`

### Logout Introspect Enabled

When enabled, the agent checks the HTTP request body to locate the Logout Request Parameter you set.

Property: `com.sun.identity.agents.config.logout.introspect.enabled`

## Logout Entry URI

Specifies the URIs to return after successful logout and subsequent authentication. The key is the web application name. The value is the URI to return.

To set a global logout entry URI for applications without other logout entry URIs defined, leave the key empty and set the value to the global logout entry URI, `/welcome.html`.

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

Property: `com.sun.identity.agents.config.logout.entry.uri`

## Access Denied URI Processing properties

### Resource Access Denied URI

Specifies the URIs of custom pages to return when access is denied. The key is the web application name. The value is the custom URI.

To set a global custom access denied URI for applications without other custom access denied URIs defined, leave the key empty and set the value to the global custom access denied URI, `/sample/accessdenied.html`.

To set a custom access denied URI for a specific application, set the key to the name of the application, and the value to the application access denied URI, such as `/myApp/accessdenied.html`.

Property: `com.sun.identity.agents.config.access.denied.uri`

## Not Enforced URI Processing properties

### Not Enforced URIs

List of URIs for which no authentication is required, and the agent does not protect access. You can use wildcards to define a pattern for a URI.

The `*` wildcard matches all characters except question mark (`?`), cannot be escaped, and spans multiple levels in a URI. Multiple forward slashes do not match a single forward slash, so `*` matches `mult/iple/dirs`, yet `mult/*/dirs` does not match `mult/dirs`.

The `-*` wildcard matches all characters except forward slash (`/`) or question mark (`?`), and cannot be escaped. As it does not match `/`, `-*` does not span multiple levels in a URI.

OpenAM does not let you mix `*` and `-*` in the same URI.

Examples include `/logout.html`, `/images/*`, `/css/-*`, and `/*.jsp?locale=*`.

Trailing forward slashes are not recognized as part of a resource name. Therefore `/images//` and `/images` are equivalent.

Property: `com.sun.identity.agents.config.notenforced.uri`

### **Invert Not Enforced URIs**

Only enforce not enforced list of URIs. In other words, enforce policy only for those URIs and patterns specified in the list.

Property: `com.sun.identity.agents.config.notenforced.uri.invert`

### **Not Enforced URIs Cache Enabled**

When enabled, the agent caches evaluation of the not enforced URI list.

Property: `com.sun.identity.agents.config.notenforced.uri.cache.enable`

### **Not Enforced URIs Cache Size**

When caching is enabled, this limits the number of not enforced URIs cached.

Property: `com.sun.identity.agents.config.notenforced.uri.cache.size`

Default: 1000

### **Refresh Session Idle Time**

When enabled, the agent reset the session idle time when granting access to a not enforced URI, prolonging the time before the user must authenticate again.

Property: `com.sun.identity.agents.config.notenforced.refresh.session.idletime`

## *Not Enforced IP Processing properties*

### **Not Enforced Client IP List**

No authentication and authorization are required for the requests coming from these client IP addresses.

Property: `com.sun.identity.agents.config.notenforced.ip`

### **Not Enforced IP Invert List**

Only enforce the not enforced list of IP addresses. In other words, enforce policy only for those client addresses and patterns specified in the list.

Property: `com.sun.identity.agents.config.notenforced.ip.invert`

### **Not Enforced IP Cache Flag**

When enabled, the agent caches evaluation of the not enforced IP list.

Property: `com.sun.identity.agents.config.notenforced.ip.cache.enable`

### Not Enforced IP Cache Size

When caching is enabled, this limits the number of not enforced addresses cached.

Property: `com.sun.identity.agents.config.notenforced.ip.cache.size`

Default: 1000

### Profile Attributes Processing properties

#### Profile Attribute Fetch Mode

When set to `HTTP_COOKIE` or `HTTP_HEADER`, profile attributes are introduced into the cookie or the headers, respectively. When set to `REQUEST_ATTRIBUTE`, profile attributes are part of the HTTP request.

Property: `com.sun.identity.agents.config.profile.attribute.fetch.mode`

#### Profile Attribute Map

Maps the profile attributes to HTTP headers for the currently authenticated user. Map Keys are LDAP attribute names, and Map Values are HTTP header names.

To populate the value of profile attribute CN under `CUSTOM-Common-Name`: enter CN in the Map Key field, and enter `CUSTOM-Common-Name` in the Corresponding Map Value field. This corresponds to `com.sun.identity.agents.config.profile.attribute.mapping[cn]=CUSTOM-Common-Name`.

In most cases, in a destination application where an HTTP header name shows up as a request header, it is prefixed by `HTTP_`, lower case letters become upper case, and hyphens (-) become underscores (\_). For example, `common-name` becomes `HTTP_COMMON_NAME`.

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

### Response Attributes Processing properties

#### Response Attribute Fetch Mode

When set to `HTTP_COOKIE` or `HTTP_HEADER`, response attributes are introduced into the cookie or the headers, respectively. When set to `REQUEST_ATTRIBUTE`, response attributes are part of the HTTP response.

Property: `com.sun.identity.agents.config.response.attribute.fetch.mode`

#### Response Attribute Map

Maps the policy response attributes to HTTP headers for the currently authenticated user. The response attribute is the attribute in the policy response to be fetched.

To populate the value of response attribute `uid` under `CUSTOM-User-Name`: enter `uid` in the Map Key field, and enter `CUSTOM-User-Name` in the Corresponding Map Value field. This corresponds to `com.sun.identity.agents.config.response.attribute.mapping[uid]=Custom-User-Name`.

In most cases, in a destination application where an HTTP header name shows up as a request header, it is prefixed by `HTTP_`, lower case letters become upper case, and hyphens (-) become underscores (\_). For example, `response-attr-one` becomes `HTTP_RESPONSE_ATTR_ONE`.

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

## Common Attributes Fetching Processing properties

### Cookie Separator Character

Specifies the separator for multiple values of the same attribute when it is set as a cookie. Default: `|` (also known as the vertical bar character).

Property: `com.sun.identity.agents.config.attribute.cookie.separator`

### Fetch Attribute Date Format

Specifies the `java.text.SimpleDateFormat` of date attribute values used when an attribute is set in an HTTP header. Default: `EEE, d MMM yyyy hh:mm:ss z`.

Property: `com.sun.identity.agents.config.attribute.date.format`

### Attribute Cookie Encode

When enabled, attribute values are URL encoded before being set as a cookie.

Property: `com.sun.identity.agents.config.attribute.cookie.encode`

## Session Attributes Processing properties

### Session Attribute Fetch Mode

When set to `HTTP_COOKIE` or `HTTP_HEADER`, session attributes are introduced into the cookie or the headers, respectively. When set to `REQUEST_ATTRIBUTE`, session attributes are part of the HTTP response.

Property: `com.sun.identity.agents.config.session.attribute.fetch.mode`

### Session Attribute Map

Maps session attributes to HTTP headers for the currently authenticated user. The session attribute is the attribute in the session to be fetched.

To populate the value of session attribute `UserToken` under `CUSTOM-userid`: enter `UserToken` in the Map Key field, and enter `CUSTOM-userid` in the Corresponding Map Value field. This corresponds to `com.sun.identity.agents.config.session.attribute.mapping[UserToken]=CUSTOM-userid`.

In most cases, in a destination application where an HTTP header name shows up as a request header, it is prefixed by `HTTP_`, lower case letters become upper case, and hyphens (-) become underscores (\_). For example, `success-url` becomes `HTTP_SUCCESS_URL`.

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

## *Privilege Attributes Processing properties*

### **Default Privileged Attribute**

Specifies the list of privileged attributes granted to all users with a valid OpenAM session, such as `AUTHENTICATED_USERS`.

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

### **Privileged Attribute Type**

Specifies the list of privileged attribute types fetched for each user.

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

### **Privileged Attributes To Lower Case**

Specifies how privileged attribute types should be converted to lower case.

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

### **Privileged Session Attribute**

Specifies the list of session property names, such as `UserToken` which hold privileged attributes for authenticated users.

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

### **Enable Privileged Attribute Mapping**

When enabled, lets you use Privileged Attribute Mapping.

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

### **Privileged Attribute Mapping**

OpenAM allows original attribute values to be mapped to other values. For example, you can map UUIDs to principal names in roles specified in a web application's deployment descriptor. For example, to map the UUID `id=employee,ou=group,o=openam` to the principal name `am_employee_role` in the deployment descriptor, set the key to `id=employee,ou=group,o=openam`, and the value to `am_employee_role`.

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

## Custom Authentication Processing properties

### Custom Authentication Handler

Specifies custom authentication handler classes for users authenticated with the application server. The key is the web application name and the value is the authentication handler class name.

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

### Custom Logout Handler

Specifies custom logout handler classes to log users out of the application server. The key is the web application name and the value is the logout handler class name.

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

### Custom Verification Handler

Specifies custom verification classes to validate user credentials with the local user repository. The key is the web application name and the value is the validation handler class name.

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

## 6.6.3. Configuring J2EE Policy Agent SSO Properties

This section covers SSO web agent properties. After creating the agent profile, you access these properties in the OpenAM console under Access Control > *Realm Name* > Agents > J2EE > *Agent Name* > SSO

### Cookie properties

#### Cookie Name

Name of the SSO Token cookie used between the OpenAM server and the agent. Default: `iPlanetDirectoryPro`.

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

Hot swap: no

### Caching properties

#### SSO Cache Enable

When enabled, the agent exposes SSO Cache through the agent SDK APIs.

Property: `com.sun.identity.agents.config.amsso.cache.enable`

## Cross Domain SSO properties

### Cross Domain SSO

Enables Cross Domain Single Sign On.

Property: `com.sun.identity.agents.config.cdsso.enable`

### CDSSO Redirect URI

Specifies a URI the agent uses to process CDSSO requests.

Property: `com.sun.identity.agents.config.cdsso.redirect.uri`

### CDSSO Servlet URL

List of URLs of the available CDSSO controllers that the agent can use for CDSSO processing. For example, `http://openam.example.com:8080/openam/cdcervlet`.

Property: `com.sun.identity.agents.config.cdsso.cdcervlet.url`

### CDSSO Clock Skew

When set to a value other than zero, specifies the clock skew in seconds that the agent accepts when determining the validity of the CDSSO authentication response assertion.

Property: `com.sun.identity.agents.config.cdsso.clock.skew`

### CDSSO Trusted ID Provider

Specifies the list of OpenAM servers or identity providers the agent trusts when evaluating CDC Liberty Responses.

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

### CDSSO Secure Enable

When enabled, the agent marks the SSO Token cookie as secure, thus the cookie is only transmitted over secure connections.

Property: `com.sun.identity.agents.config.cdsso.secure.enable`

### CDSSO Domain List

List of domains, such as `.example.com`, in which cookies have to be set in CDSSO.

Property: `com.sun.identity.agents.config.cdsso.domain`

## Cookie Reset properties

### Cookie Reset

When enabled, agent resets cookies in the response before redirecting to authentication.

Property: `com.sun.identity.agents.config.cookie.reset.enable`

### Cookie Reset Name List

List of cookies to reset if Cookie Reset is enabled.

Property: `com.sun.identity.agents.config.cookie.reset.name`

### Cookie Reset Domain Map

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

Property: `com.sun.identity.agents.config.cookie.reset.domain`

### Cookie Reset Path Map

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

Property: `com.sun.identity.agents.config.cookie.reset.path`

## 6.6.4. Configuring J2EE Policy Agent OpenAM Services Properties

This section covers OpenAM services web agent properties. After creating the agent profile, you access these properties in the OpenAM console under Access Control > *Realm Name* > Agents > J2EE > *Agent Name* > OpenAM Services.

## Login URL properties

### OpenAM Login URL

OpenAM login page URL, such as `http://openam.example.com:8080/openam/UI/Login`, to which the agent redirects incoming users without sufficient credentials so then can authenticate.

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

### OpenAM Conditional Login URL (Not yet in OpenAM console)

To conditionally redirect users based on the incoming request URL, set this property.

This takes the incoming request domain to match, a vertical bar ( | ), and then a comma-separated list of URLs to which to redirect incoming users.

If the domain before the vertical bar matches an incoming request URL, then the policy agent uses the list of URLs to determine how to redirect the user-agent. If the global property FQDN Check (`com.sun.identity.agents.config.fqdn.check.enable`) is enabled for the policy agent, then the policy agent iterates through the list until it finds an appropriate redirect URL that matches the FQDN check. Otherwise, the policy agent redirects the user-agent to the first URL in the list.

Property: `com.sun.identity.agents.config.conditional.login.url`

Examples: `com.sun.identity.agents.config.conditional.login.url[0]= login.example.com|http://openam1.example.com/openam/UI/Login, http://openam2.example.com/openam/UI/Login, com.sun.identity.agents.config.conditional.login.url[1]= signin.example.com|http://openam3.example.com/openam/UI/Login, http://openam4.example.com/openam/UI/Login`

If CDSSO is enabled for the policy agent, then this property takes CDSSO Servlet URLs for its values (`com.sun.identity.agents.config.cdsso.cdcservlet.url`), rather than OpenAM login URLs.

CDSSO examples: `com.sun.identity.agents.config.conditional.login.url[0]= login.example.com|http://openam1.example.com/openam/cdcservlet, http://openam2.example.com/openam/cdcservlet, com.sun.identity.agents.config.conditional.login.url[1]= signin.example.com|http://openam3.example.com/openam/cdcservlet, http://openam4.example.com/openam/cdcservlet`

## Login URL Prioritized

When enabled, OpenAM uses the priority defined in the OpenAM Login URL list as the priority for Login and CDSSO URLs when handling failover.

Property: `com.sun.identity.agents.config.login.url.prioritized`

## Login URL Probe

When enabled, OpenAM checks the availability of OpenAM Login URLs before redirecting to them.

Property: `com.sun.identity.agents.config.login.url.probe.enabled`

## Login URL Probe Timeout

Timeout period in milliseconds for OpenAM to determine whether to failover between Login URLs when Login URL Probe is enabled.

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

Default: 2000

## Logout URL properties

### OpenAM Logout URL

OpenAM logout page URLs, such as `http://openam.example.com:8080/openam/UI/Logout`. The user is logged out of the OpenAM session when accessing these URLs.

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

### OpenAM Conditional Logout URL (Not yet in OpenAM console)

The values take the incoming request URL to match and a comma-separated list of URLs to which to redirect users logging out.

Property: `com.sun.identity.agents.config.conditional.logout.url`

Example: `com.sun.identity.agents.config.conditional.logout.url[0]= logout.example.com|http://openam1.example.com/openam/UI/Logout, http://openam2.example.com/openam/UI/Logout`

### Logout URL Prioritized

When enabled, OpenAM uses the priority defined in the OpenAM Logout URL list as the priority for Logout URLs when handling failover.

Property: `com.sun.identity.agents.config.logout.url.prioritized`

### Logout URL Probe

When enabled, OpenAM checks the availability of OpenAM Logout URLs before redirecting to them.

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

### Logout URL Probe Timeout

Timeout period in milliseconds for OpenAM to determine whether to failover between Logout URLs when Logout URL Probe is enabled.

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

Default: 2000

## *Authentication Service properties*

### OpenAM Authentication Service Protocol

Specifies the protocol used by the OpenAM authentication service.

Property: `com.iplanet.am.server.protocol`

Hot swap: no

### OpenAM Authentication Service Host Name

Specifies the OpenAM authentication service host name.

Property: `com.iplanet.am.server.host`

Hot swap: no

### OpenAM Authentication Service Port

Specifies the OpenAM authentication service port number.

Property: `com.iplanet.am.server.port`

Hot swap: no

### *Policy Client Service properties*

#### Enable Policy Notifications

When enabled, OpenAM sends notification about changes to policy.

Property: `com.sun.identity.agents.notification.enabled`

Hot swap: no

#### Policy Client Polling Interval

Specifies the time in minutes after which the policy cache is refreshed.

Property: `com.sun.identity.agents.polling.interval`

Default: 3

Hot swap: no

#### Policy Client Cache Mode

Set to cache mode subtree when only a small number of policy rules are defined. For large numbers of policy rules, set to self.

Property: `com.sun.identity.policy.client.cacheMode`

Default: self

Hot swap: no

#### Policy Client Boolean Action Values

Specifies the values, such as `allow` and `deny`, that are associated with boolean policy decisions.

Default: `iPlanetAMWebAgentService|GET|allow|deny:iPlanetAMWebAgentService|POST|allow|deny`

Property: `com.sun.identity.policy.client.booleanActionValues`

Hot swap: no

## Policy Client Resource Comparators

Specifies the comparators used for service names in policy.

Default: `serviceType=iPlanetAMWebAgentService| class=com.sun.identity.policy.plugins.HttpURLResourceName|wildcard=*| delimiter=/|caseSensitive=false`

Property: `com.sun.identity.policy.client.resourceComparators`

Hot swap: no

## Policy Client Clock Skew

Time in seconds used adjust time difference between agent system and OpenAM. Clock skew in seconds = AgentTime - OpenAMServerTime.

Default: 10.

Property: `com.sun.identity.policy.client.clockSkew`

Hot swap: no

## URL Policy Env GET Parameters

Specifies the list of HTTP GET request parameters whose names and values the agents sets in the environment map for URL policy evaluation by the OpenAM server.

Property: `com.sun.identity.agents.config.policy.env.get.param`

## URL Policy Env POST Parameters

Specifies the list of HTTP POST request parameters whose names and values the agents sets in the environment map for URL policy evaluation by the OpenAM server.

Property: `com.sun.identity.agents.config.policy.env.post.param`

## URL Policy Env jsession Parameters

Specifies the list of HTTP session attributes whose names and values the agents sets in the environment map for URL policy evaluation by the OpenAM server.

Property: `com.sun.identity.agents.config.policy.env.jsession.param`

## Use HTTP-Redirect for composite advice

When enabled, the remote policy client is configured to use HTTP-Redirect instead of HTTP-POST for composite advice.

Property: `com.sun.identity.agents.config.policy.advice.use.redirect`

## *User Data Cache Service properties*

### **Enable Notification of User Data Caches**

When enabled, receive notification from OpenAM to update user management data caches.

Property: `com.sun.identity.idm.remote.notification.enabled`

Hot swap: no

### **User Data Cache Polling Time**

If notifications are not enabled and set to a value other than zero, specifies the time in minutes after which the agent polls to update cached user management data.

Property: `com.iplanet.am.sdk.remote.pollingTime`

Default: 1

Hot swap: no

### **Enable Notification of Service Data Caches**

When enabled, receive notification from OpenAM to update service configuration data caches.

Property: `com.sun.identity.sm.notification.enabled`

Hot swap: no

### **Service Data Cache Time**

If notifications are not enabled and set to a value other than zero, specifies the time in minutes after which the agent polls to update cached service configuration data.

Property: `com.sun.identity.sm.cacheTime`

Default: 1

Hot swap: no

## *Session Client Service properties*

### **Enable Client Polling**

When enabled, the session client polls to update the session cache rather than relying on notifications from OpenAM.

Property: `com.iplanet.am.session.client.polling.enable`

Hot swap: no

### Client Polling Period

Specifies the time in seconds after which the session client requests an update from OpenAM for cached session information.

Property: `com.iplanet.am.session.client.polling.period`

Default: 180

Hot swap: no

## 6.6.5. Configuring J2EE Policy Agent Miscellaneous Properties

This section covers miscellaneous web agent properties. After creating the agent profile, you access these properties in the OpenAM console under Access Control > *Realm Name* > Agents > J2EE > *Agent Name* > Miscellaneous.

### *Locale properties*

#### Locale Language

The default language for the agent.

Property: `com.sun.identity.agents.config.locale.language`

Hot swap: no

#### Locale Country

The default country for the agent.

Property: `com.sun.identity.agents.config.locale.country`

Hot swap: no

### *Port Check Processing properties*

#### Port Check Enable

When enabled, activate port checking, correcting requests on the wrong port.

Property: `com.sun.identity.agents.config.port.check.enable`

#### Port Check File

Specifies the name of the file containing the content to handle requests on the wrong port when port checking is enabled.

Property: `com.sun.identity.agents.config.port.check.file`

### Port Check Setting

Specifies which ports correspond to which protocols. The agent uses the map when handling requests with invalid port numbers during port checking.

Property: `com.sun.identity.agents.config.port.check.setting`

### *Bypass Principal List properties*

#### **Bypass Principal List**

Specifies a list of principals the agent bypasses for authentication and search purposes, such as `guest` or `testuser`.

Property: `com.sun.identity.agents.config.bypass.principal`

### *Agent Password Encryptor properties*

#### **Encryption Provider**

Specifies the agent's encryption provider class.

Default: `com.ipplanet.services.util.JCEEncryption`

Property: `com.ipplanet.security.encryptor`

Hot swap: no

### *Ignore Path Info properties*

#### **Ignore Path Info in Request URL**

When enabled, strip path info from the request URL while doing the Not Enforced List check, and URL policy evaluation. This is designed to prevent a user from accessing a URI by appending the matching pattern in the policy or not enforced list.

For example, if the not enforced list includes `/*.gif`, then stripping path info from the request URL prevents access to `http://host/index.html` by using `http://host/index.html?hack.gif`.

Property: `com.sun.identity.agents.config.ignore.path.info`

### *Deprecated Client Browser User Agent Properties*

#### **Goto Parameter Name**

Property used only when CDSSO is enabled. Only change the default value, `goto` when the login URL has a landing page specified such as, `com.sun.identity.agents.config.cdssso.cdcservlet.url =`

`http://openam.example.com:8080/openam/cdcervlet?goto= http://www.example.com/landing.jsp`. The agent uses this parameter to append the original request URL to this cdcserlet URL. The landing page consumes this parameter to redirect to the original URL.

As an example, if you set this value to `goto2`, then the complete URL sent for authentication is `http://openam.example.com:8080/openam/cdcervlet?goto= http://www.example.com/landing.jsp?goto2=http://www.example.com/original.jsp`.

Property: `com.sun.identity.agents.config.redirect.param`

### Legacy User Agent Support Enable

When enabled, provide support for legacy browsers.

Property: `com.sun.identity.agents.config.legacy.support.enable`

### Legacy User Agent List

List of header values that identify legacy browsers. Entries can use the wildcard character, `*`.

Property: `com.sun.identity.agents.config.legacy.user.agent`

### Legacy User Agent Redirect URI

Specifies a URI the agent uses to redirect legacy user agent requests.

Property: `com.sun.identity.agents.config.legacy.redirect.uri`

## 6.6.6. Configuring J2EE Policy Agent Advanced Properties

This section covers advanced web agent properties. After creating the agent profile, you access these properties in the OpenAM console under Access Control > *Realm Name* > Agents > J2EE > *Agent Name* > Advanced.

### *Client Identification properties*

If the agent is behind a proxy or load balancer, then the agent can get client IP and host name values from the proxy or load balancer. For proxies and load balancer that support providing the client IP and host name in HTTP headers, you can use the following properties.

When multiple proxies are load balancers sit in the request path, the header values can include a comma-separated list of values with the first value representing the client, as in `client,next-proxy ,first-proxy`.

### Client IP Address Header

HTTP header name that holds the IP address of the client.

Property: `com.sun.identity.agents.config.client.ip.header`

## Client Hostname Header

HTTP header name that holds the hostname of the client.

Property: `com.sun.identity.agents.config.client.hostname.header`

## Web Service Processing properties

### Web Service Enable

Enable web service processing.

Property: `com.sun.identity.agents.config.webservice.enable`

### Web Service End Points

Specifies a list of web application end points that represent web services.

Property: `com.sun.identity.agents.config.webservice.endpoint`

### Web Service Process GET Enable

When enabled, the agent processes HTTP GET requests for web service endpoints.

Property: `com.sun.identity.agents.config.webservice.process.get.enable`

### Web Service Authenticator

Specifies a class implementing `com.sun.identity.agents.filter.IWebServiceAuthenticator`, used to authenticate web service requests.

Property: `com.sun.identity.agents.config.webservice.responseprocessor`

### Web Service Response Processor

Specifies a class implementing `com.sun.identity.agents.filter.IWebServiceResponseProcessor`, used to process web service responses.

Property: `com.sun.identity.agents.config.webservice.responseprocessor`

### Web Service Internal Error Content File

Specifies a file the agent uses to generate an internal error fault for the client application.

Property: `com.sun.identity.agents.config.webservice.internalerror.content`

### Web Service Authorization Error Content File

Specifies a file the agent uses to generate an authorization error fault for the client application.

Property: `com.sun.identity.agents.config.webservice.autherror.content`

## *Alternate Agent URL properties*

### **Alternative Agent Host Name**

Specifies the host name of the agent protected server to show to client browsers, rather than the actual host name.

Property: `com.sun.identity.agents.config.agent.host`

### **Alternative Agent Port Name**

Specifies the port number of the agent protected server to show to client browsers, rather than the actual port number.

Property: `com.sun.identity.agents.config.agent.port`

### **Alternative Agent Protocol**

Specifies the protocol used to contact the agent from the browser client browsers, rather than the actual protocol used by the server. Either `http` or `https`.

Property: `com.sun.identity.agents.config.agent.protocol`

## *JBoss Application Server properties*

### **WebAuthentication Available**

When enabled, allow programmatic authentication with the JBoss container using the WebAuthentication feature. This feature works only with JBoss 4.2.2 to 7 when the `J2EE_POLICY` or `ALL` filter mode is in use.

Property: `com.sun.identity.agents.config.jboss.webauth.available`

## *Cross Site Scripting Detection properties*

### **Possible XSS code elements**

Specifies strings that, when found in the request, cause the agent to redirect the client to an error page.

Property: `com.sun.identity.agents.config.xss.code.elements`

### **XSS detection redirect URI**

Maps applications to URIs of customized pages to which to redirect clients upon detection of XSS code elements.

For example, to redirect clients of MyApp to `/myapp/error.html`, enter MyApp as the Map Key and `/myapp/error.html` as the Corresponding Map Value.

Property: `com.sun.identity.agents.config.xss.redirect.uri`

## Post Data Preservation properties

### POST Data Preservation

Enables HTTP POST data preservation, storing POST data before redirecting the browser to the login screen, and then autosubmitting the same POST after successful authentication to the original URL.

Property: `com.sun.identity.agents.config.postdata.preserve.enable`

### Missing PDP entry URI

Specifies a list of application-specific URIs if the referenced Post Data Preservation entry cannot be found in the local cache because it has exceeded its POST entry TTL. Either the agent redirects to a URI in this list, or it shows an HTTP 403 Forbidden error.

Property: `com.sun.identity.agents.config.postdata.preserve.cache.noentry.url`

### POST entry TTL

POST data storage lifetime in milliseconds. Default: 300000.

Property: `com.sun.identity.agents.config.postdata.preserve.cache.entry.ttl`

### PDP Sticky session mode

Specifies whether to create a cookie, or to append a query string to the URL to assist with sticky load balancing.

Property: `com.sun.identity.agents.config.postdata.preserve.stickysession.mode`

### PDP Sticky session key-value

Specifies the key-value pair for sticky session mode. For example, a setting of `lb=myserver` either sets an `lb` cookie with `myserver` value, or adds `lb=myserver` to the URL query string.

Property: `com.sun.identity.agents.config.postdata.preserve.stickysession.value`

## Custom properties

### Custom Properties

Additional properties to augment the set of properties supported by agent. Such properties take the following forms.

- `customproperty=custom-value1`
- `customlist[0]=customlist-value-0`

- `customlist[1]=customlist-value-1`
- `custommap[key1]=custommap-value-1`
- `custommap[key2]=custommap-value-2`

Property: `com.sun.identity.agents.config.freeformproperties`

## 6.7. Configuring Web Service Provider Policy Agents

This section covers Web Service Provider (WSP) properties. WSPs both validate incoming web service requests from Web Service Clients (WSC), and also secure outgoing responses sent back to WSCs.

After creating a WSP profile, you access WSP properties in the OpenAM console under Access Control > *Realm Name* > Agents > Web Service Provider > *Agent Name*.

### *General properties*

#### **Group**

For assigning the agent to a previously configured agent group in order to inherit selected properties from the group.

#### **Password**

Agent password used when creating the password file and when installing the agent.

#### **Status**

Status of the agent configuration.

#### **Universal Identifier**

OpenAM identifier for the agent configuration.

### *Security properties*

#### **Security Mechanism**

Specifies the mechanisms allowed to validate the web service request.

#### **Authentication Chain**

Specifies which OpenAM authentication chain consumes the credentials from the web service request to authenticate the WSC.

#### **Token Conversion Type**

Specifies how to covert the incoming token before issuing requests to other WSPs.

### **Preserve Security Headers in Message**

Yes means the agent preserves SOAP security headers from the request for subsequent processing.

### **Detect Message Replay**

Yes means the agent checks whether the request is a replay of an earlier request, and if so, rejects the request.

### **Detect User Token Replay**

Yes means the agent checks whether the user token is a replay from an earlier requests, and if so, rejects the request.

### **Private Key Type**

Specifies the type of key, such as `PublicKey`, used to verify the request signature.

### **Liberty Service Type URN**

Specifies the Universal Resource Name for the Liberty service type used for lookups.

### **DNS Claim**

Specifies a Uniform Resource Identifier shared by the WSP and WSC.

### **Credential for User Token**

Specifies the user name and password credentials compared with the user name security token in a request.

## *SAML Configuration properties*

### **SAML Attribute Mapping**

Maps SAML attribute names from the incoming request to attribute names as retrieved from the SSOToken or the identity repository, used to have the Security Token Service generate an appropriate SAML assertion.

### **SAML NameID Mapper Plugin**

Specifies the class name of a plugin used to perform SAML account mapping.

### **SAML Attributes Namespace**

Identifies the attribute name space used when generating SAML assertions.

### **Include Memberships**

Yes means the agent includes the principal's membership as a SAML attribute.

## *Signing and Encryption properties*

### **Is Request Signature Verified**

Yes means verify signatures in requests.

### **Is Response Signed Enabled**

Yes means the agent signs the specified parts of the response with its x509 certificate.

### **Signing Reference Type**

Specifies how the x509 certificate used to sign responses is referenced in the response.

### **Is Request Decrypted**

Yes means do decrypt the specified parts of incoming requests.

### **Is Response Encrypted**

Yes means do encrypt the outgoing response.

### **Encryption Algorithm**

Specifies whether to use Advanced Encryption Standard, corresponding to an Encryption Strength of 128, 192, or 256, or to use Triple DES with a key length of 0, 112, or 168.

### **Encryption Strength**

Specifies the key length used for encryption.

## *Key Store properties*

### **Public Key Alias of Web Service Client**

Specifies the alias of the certificate in the key store used to verify request signatures and encrypt responses.

### **Private Key Alias**

Specifies the alias of the certificate in the key store used to sign responses and decrypt requests.

### **Key Store Usage**

If you use your own, custom key store, specify how to access it here.

## *End Points properties*

### **Web Service Security Proxy End Point**

If the WSC sends requests through a web service proxy, specify that as the end point here.

## Web Service End Point

Specifies the end point to which the WSC sends requests.

## *Kerberos Configuration properties*

### **Kerberos Domain Server**

Specifies the fully qualified domain name of the Kerberos Distribution Center service.

### **Kerberos Domain**

Specifies the Kerberos Distribution Center domain name. For Windows environments this is the domain controller domain name.

### **Kerberos Service Principal**

Specifies the Kerberos principal used by OpenAM, using the form `HTTP/openam-fqdn@krb-domain`, where *openam-fqdn* is the fully qualified domain name for OpenAM, and *krb-domain* is the Kerberos Domain.

### **Kerberos Key Tab File**

Specifies the Kerberos keytab file using the form `openam-host.HTTP.keytab`, where *openam-host* is the host name for OpenAM.

### **Verify Kerberos Signature**

Yes means the agent signs the Kerberos token.

## 6.8. Configuring Web Service Client Policy Agents

This section covers Web Service Client (WSC) properties. WSCs both secure outgoing requests sent to Web Service Providers (WSP), and also validate incoming from WSPs.

After creating a WSC profile, you access WSC properties in the OpenAM console under Access Control > *Realm Name* > Agents > Web Service Client > *Agent Name*.

### *General properties*

#### **Group**

For assigning the agent to a previously configured agent group in order to inherit selected properties from the group.

#### **Password**

Agent password used when creating the password file and when installing the agent.

**Status**

Status of the agent configuration.

**Universal Identifier**

OpenAM identifier for the agent configuration.

*Security properties***Security Mechanism**

Specifies the mechanism used to secure web service requests.

**STS Configuration**

Specifies the agent used to secure requests to the Security Token Service. Associated with the STSSecurity Security Mechanism.

**Discovery Configuration**

Specifies the agent used to secure requests to the Discovery Service. Associated with the LibertyDiscoverySecurity Security Mechanism.

**User Authentication Required**

Yes means users must authenticate to access the WSC's protected page.

**Preserve Security Headers in Message**

Yes means the agent preserves SOAP security headers in the request for subsequent processing.

**User Pass Through Security Token**

Yes means the agent passes along the Security Token from the Subject, rather than generating a token or requesting it from the Security Token Service.

**Liberty Service Type URN**

Specifies the Universal Resource Name for the Liberty service type used for lookups.

**Credential for User Token**

Specifies the user name and password credentials shared with the WSP and used to generate a Username Security Token.

**DNS Claim**

Specifies a Uniform Resource Identifier shared by the WSP and WSC.

## *SAML Configuration properties*

### **SAML Attribute Mapping**

Maps SAML attribute names from the outgoing request to attribute names as retrieved from the SSO Token or the identity repository.

### **SAML NameID Mapper Plugin**

Specifies the class name of a plugin used to perform SAML account mapping.

### **SAML Attributes Namespace**

Identifies the attribute name space used when generating SAML assertions.

### **Include Memberships**

Yes means the agent includes the principal's membership as a SAML attribute.

## *Signing and Encryption properties*

### **Is Request Signed Enabled**

Yes means the agent signs the specified parts of the request with its x509 certificate.

### **Signing Reference Type**

Specifies how the x509 certificate used to sign requests is referenced in the request.

### **Is Response Signature Verified**

Yes means verify signatures in responses.

### **Is Request Encryption Enabled**

Yes means do encrypt the specified parts of outgoing requests.

### **Encryption Algorithm**

Specifies whether to use Advanced Encryption Standard, corresponding to an Encryption Strength of 128, 192, or 256, or to use Triple DES with a key length of 0, 112, or 168.

### **Encryption Strength**

Specifies the key length used for encryption.

### **Is Response Decrypted**

Yes means do decrypt the incoming response.

## *Key Store properties*

### **Public Key Alias of Web Service Provider**

Specifies the alias of the certificate in the key store used to sign requests and decrypt responses.

### **Private Key Alias**

Specifies the alias of the certificate in the key store used to verify response signatures and encrypt requests.

### **Key Store Usage**

If you use your own, custom key store, specify how to access it here.

## *End Points properties*

### **Web Service Security Proxy End Point**

If the WSC sends requests through a web service proxy, specify that as the end point here.

### **Web Service End Point**

Specifies the end point to which the WSC sends requests.

## *Kerberos Configuration properties*

### **Kerberos Domain Server**

Specifies the fully qualified domain name of the Kerberos Distribution Center service.

### **Kerberos Domain**

Specifies the Kerberos Distribution Center domain name. For Windows environments this is the domain controller domain name.

### **Kerberos Service Principal**

Specifies the Kerberos principal used by OpenAM, using the form `HTTP/openam-fqdn@krb-domain`, where `openam-fqdn` is the fully qualified domain name for OpenAM, and `krb-domain` is the Kerberos Domain.

### **Kerberos Ticket Cache Directory**

Specifies the directory in which Kerberos Ticket Granting Tickets (TGT) are cached. The `kinit` command stores the TGT from the KDC here.

## 6.9. Configuring Security Token Service Client Policy Agents

This section covers Security Token Service (STS) Client properties. STS clients both secure outgoing requests to trust authorities, and also validate incoming requests from trust authorities. You can configure STS clients to work with OpenAM's Security Token Service and with its Discovery Service.

After creating an STS Client profile, you access STS Client properties in the OpenAM console under Access Control > *Realm Name* > Agents > STS Client > *Agent Name*.

## *General properties*

### **Group**

For assigning the agent to a previously configured agent group in order to inherit selected properties from the group.

### **Password**

Agent password used when creating the password file and when installing the agent.

### **Status**

Status of the agent configuration.

### **WS-Trust Version**

Specifies whether to use WS-Trust 1.3 or 1.0.

### **Universal Identifier**

OpenAM identifier for the agent configuration.

## *Security properties*

### **Security Mechanism**

Specifies the mechanism used to secure the STS request.

### **STS Configuration**

Specifies the STS Client agent profile to use if the security mechanism is STS Security.

### **Preserve Security Headers in Message**

Yes means the agent preserves SOAP security headers for subsequent processing.

### **Credential for User Token**

Specifies the user name and password credentials the agent uses to generate a Username security token.

**Requested Key Type**

Specifies the type of key, such as `PublicKey`, used to encrypt responses.

**Requested Claims**

Specifies the Uniform Resource Identifiers for the claims to be represented in the Security Token.

**DNS Claim**

Specifies a Uniform Resource Identifier shared by the agent and the WSC.

*SAML Configuration properties***SAML Attribute Mapping**

Maps SAML attribute names from the incoming request to attribute names as retrieved from the SSO Token or the identity repository, used to have the Security Token Service generate an appropriate SAML assertion.

**SAML NameID Mapper Plugin**

Specifies the class name of a plugin used to perform SAML account mapping.

**SAML Attributes Namespace**

Identifies the attribute name space used when generating SAML assertions.

**Include Memberships**

Yes means the agent includes the principal's membership as a SAML attribute.

*Signing and Encryption properties***Is Response Signature Verified**

Yes means verify signatures in responses.

**Is Request Signed Enabled**

Yes means the agent signs the specified parts of the request with its x509 certificate.

**Signing Reference Type**

Specifies how the x509 certificate used to sign requests is referenced in the request.

**Is Request Encryption Enabled**

Yes means do encrypt the specified parts of requests.

## **Is Response Decrypted**

Yes means do decrypt the response.

## **Encryption Algorithm**

Specifies whether to use Advanced Encryption Standard, corresponding to an Encryption Strength of 128, 192, or 256, or to use Triple DES with a key length of 0, 112, or 168.

## **Encryption Strength**

Specifies the key length used for encryption.

## *Key Store properties*

### **Public Key Alias of Web Service Provider**

Specifies the alias of the certificate in the key store used to verify response signatures and encrypt requests.

### **Private Key Alias**

Specifies the alias of the certificate in the key store used to sign requests and decrypt responses.

### **Key Store Usage**

If you use your own, custom key store, specify how to access it here.

## *End Points properties*

### **Security Token Service End Point**

Specifies the URL to the Security Token Service end point.

### **Security Token Service MEX End Point**

Specifies the URL to the Security Token Service message exchange end point.

## *Kerberos Configuration properties*

### **Kerberos Domain Server**

Specifies the fully qualified domain name of the Kerberos Distribution Center service.

### **Kerberos Domain**

Specifies the Kerberos Distribution Center domain name. For Windows environments this is the domain controller domain name.

### Kerberos Service Principal

Specifies the Kerberos principal used by OpenAM, using the form `HTTP/openam-fqdn@krb-domain`, where `openam-fqdn` is the fully qualified domain name for OpenAM, and `krb-domain` is the Kerberos Domain.

### Kerberos Ticket Cache Directory

Specifies the directory in which Kerberos Ticket Granting Tickets (TGT) are cached. The `kinit` command stores the TGT from the KDC here.

## 6.10. Configuring Version 2.2 Policy Agents

This section covers version 2.2 agent properties. Version 2.2 agents store their configurations locally, with a user name, password combination used to connect to OpenAM.

After creating the agent profile, you access agent properties in the OpenAM console under Access Control > *Realm Name* > Agents > 2.2 Agents > *Agent Name*.

### Password

Specifies the password the agent uses to connect to OpenAM.

### Status

Specifies whether the agent profile is active, and so can be used.

### Description

Specifies a short description for the agent.

### Agent Key Value(s)

Additional key-value pairs that OpenAM uses to receive agent requests concerning credential assertions.

OpenAM currently supports one property, `agentRootURL=protocol://host:port/` where the key is case-sensitive.

## 6.11. Configuring OAuth 2.0 & OpenID Connect 1.0 Clients

When you want to register an OAuth 2.0 client with OpenAM as the OAuth 2.0 authorization server, or register an OpenID Connect 1.0 client through OpenAM console, then create an OAuth 2.0 Client agent profile. After creating the agent profile, you can further configure the properties in the OpenAM console under Access Control > *Realm Name* > Agents > OAuth 2.0 Client > *Client Name*.

The topmost configuration fields are for both OAuth 2.0 and OpenID Connect 1.0, whereas others are specifically for OpenID Connect 1.0.

- Common Client Configuration
- OpenID Connect 1.0 Client Configuration

### *Common Client Configuration*

The following configuration fields are common to OAuth 2.0 and OpenID Connect 1.0 clients.

#### **Group**

Set this if you have configured an OAuth 2.0 Client agent group.

#### **Status**

Whether the client profile is active for use.

#### **Client password**

The client password as described by RFC 6749 in the section, Client Password.

#### **Client type**

Confidential clients can maintain confidentiality of their credentials. Public clients cannot.

A web application running on a server where its credentials are protected is an example of a confidential client.

A JavaScript client running in a browser is an example of a public client.

#### **Redirection URIs**

Specify client redirection endpoint URIs as described by RFC 6749 in the section, Redirection Endpoint. OpenAM's OAuth 2.0 authorization service redirects the the resource owner's user-agent back to this endpoint during the authorization code grant process. If your client has more than one redirection URI, then it must specify the redirection URI to use in the authorization request.

Redirection URIs are required for OpenID Connect 1.0 clients.

#### **Scopes**

Specify scopes in *scope* or *scope|locale|localized description* format. These scopes are to be presented to the resource owner when the resource owner is asked to authorize client access to protected resources.

#### **Display name**

Specify a client name to display to the resource owner when the resource owner is asked to authorize client access to protected resources. Valid formats include *name* or *locale|localized name*.

## Display description

Specify a client description to display to the resource owner when the resource owner is asked to authorize client access to protected resources. Valid formats include *description* or *locale|localized description*.

## Default Scope(s)

Specify scopes in *scope* or *scope|locale|localized description* format. These scopes are set automatically when tokens are issued.

## OpenID Connect 1.0 Client Configuration

The following optional configuration fields are for OpenID Connect 1.0 clients.

### ID Token Signed Response Algorithm

Algorithm that the ID Token for this client must be signed with

Default: *HmacSHA256*

Valid values are listed in *JSON Web Algorithms (JWA): "alg" (Algorithm) Header Parameter Values for JWS*. OpenAM supports *HmacSHA256*, *HmacSHA384*, and *HmacSHA512*.

### Post Logout Redirect URI

URI to which to redirect the user-agent after the client logout process

### The access token used to update the client

The *registration\_access\_token* value that you provide when registering the client, and then subsequently when reading or updating the client profile.

### URI of the Client Application

The relying party (client) URI to which the OpenID Connect Provider sends session changed notification messages using the HTML 5 *postMessage* API.

## 6.12. Configuring Agent Authenticators

An *agent authenticator* has read-only access to multiple agent profiles defined in the same realm, typically allowing an agent to read web service agent profiles.

After creating the agent profile, you access agent properties in the OpenAM console under Access Control > *Realm Name* > Agents > Agent Authenticator > *Agent Name*.

### Password

Specifies the password the agent uses to connect to OpenAM.

## Status

Specifies whether the agent profile is active, and so can be used.

## Agent Profiles allow to Read

Specifies which agent profiles in the realm the agent authenticator can read.

## Agent Root URL for CDSSO

Specifies the list of agent root URLs for CDSSO. The valid value is in the format *protocol://hostname:port/* where *protocol* represents the protocol used, such as *http* or *https*, *hostname* represents the host name of the system where the agent resides, and *port* represents the port number on which the agent is installed. The slash following the port number is required.

If your agent system also has virtual host names, add URLs with the virtual host names to this list as well. OpenAM checks that *goto* URLs match one of the agent root URLs for CDSSO.

## Chapter 7

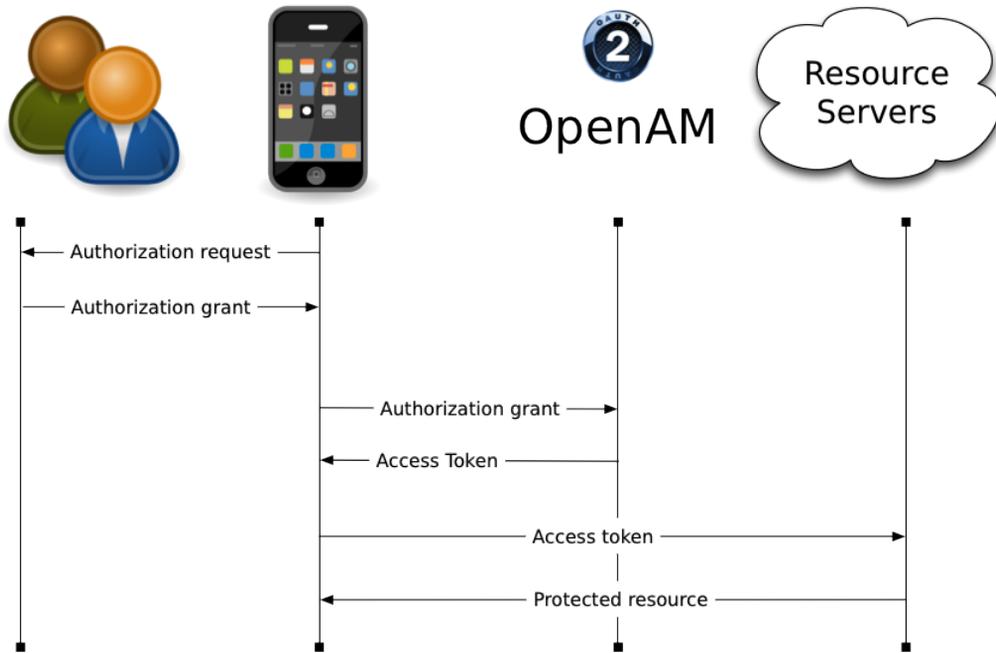
# Working with Mobile Devices & Applications

When building applications that run on mobile devices, you can use the same OpenAM service that you also use for access management in your web, cloud, and other applications. OpenAM has features that make it particularly well suited for the mobile world, too.

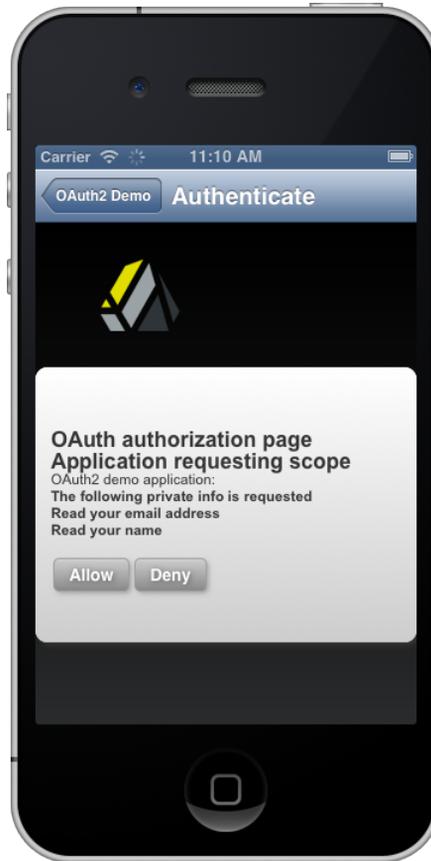
## 7.1. Simplifying Access on Mobile Devices

On many mobile devices, users want to avoid repeatedly entering credentials such as an email address or user name and a password. They do not want new credentials to manage for every application they try. They do not want to share their credentials across applications. Instead users want single sign-on with few identity providers. They want to authorize access for applications rather than share their credentials.

OpenAM supports emerging standards including OAuth 2.0 and OpenID Connect 1.0. After you register your application as an OAuth or OpenID Connect client with OpenAM, your application can then redirect a user to OpenAM to authenticate and to authorize access to resources that the user owns, such as profile data. Your application gets an access token that can be used later to gain authorized access without requiring the user to authenticate again. OpenID Connect extends OAuth, standardizing how client applications discover and register with identity providers, and also defining how applications can manage user sessions and handle logout when they no longer want to authorize access.



An OAuth 2.0 client application can thus make simplify the user experience on the phone to authorizing access.



In addition to serving as an identity provider, OpenAM can also function as an OAuth 2.0 client, protecting access to resources within your control based on authorization granted by an identity provider who users already know and use, such as Facebook, Google, MSN and others. OpenAM's built in authorization policy management makes it straightforward to integrate this capability into your applications.

The OAuth and OpenID Connect standards specify REST interfaces, making them essentially programming language-independent and accessible for web applications and mobile applications alike.

OpenAM also supports Open Authentication architecture with the OATH module mentioned in the next section.

## 7.2. Protecting Access for Mobile Users

You must give users access to your organization's resources while they are on the go. At the same time you must manage risk. OpenAM supports risk-based adaptive authentication, device fingerprints, one-time passwords and other multi-factor authentication capabilities that help you do both. As OpenAM handles authentication through plugin modules that you can chain, your OpenAM service can meet a variety of requirements.

OpenAM's Adaptive Risk authentication module lets you add risk assessment to any authentication module chain, dynamically requiring stronger authentication when circumstances require it (new location, ancient last login time, new device, new IP address, specific application, and so forth). You can add the Device Print module to an authentication chain to fingerprint users' devices for additional risk assessment, making it easier to handle sign-on when users bring their own devices.

### AdaptiveRisk - Properties

Save Reset Back to Authentication

(3 Item(s))

Add Remove Reorder

<input checked="" type="checkbox"/>	Instance	Criteria	Options
<input type="checkbox"/>	LDAP	REQUIRED	
<input type="checkbox"/>	AdaptiveRisk	SUFFICIENT	
<input type="checkbox"/>	HOTP	REQUIRED	

OpenAM also lets you decide exactly what stronger authentication means in your situation. You can for example add multi-factor authentication involving mobile devices using OpenAM's OATH and HOTP modules either to use a one-time password soft token generated on a device, or to send a one-time password in a text message to a mobile phone.

In addition to capabilities supporting new applications, OpenAM integrates well with existing systems needed by users on the move. Whether users are authenticating from a mobile device through a gateway using an MSISDN, starting single sign-on by logging on to a laptop, or connecting to a VPN with certificate based authentication, OpenAM has an authentication module for that.

---

**New Module Instance** OK Cancel

\* Indicates required field

\* Name:

\* Type:

- Active Directory
- Adaptive Risk
- Anonymous
- Certificate
- Data Store
- Device Print
- Federation
- HOTP
- HTTP Basic
- JDBC
- LDAP
- Membership
- MSISDN
- OATH
- OAuth 2.0
- Persistent Cookie
- RADIUS
- SAE
- SecurID
- Windows Desktop SSO
- Windows NT
- WSSAuth

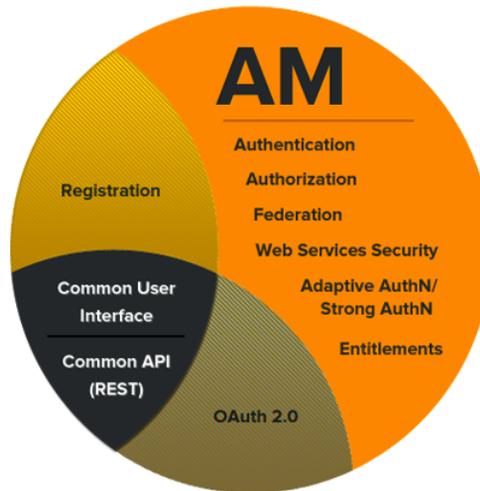
All of these capabilities are available with OpenAM out of the box.

## 7.3. Simplifying Access with REST APIs

Representational State Transfer (REST) is a architectural style designed in parallel with HTTP. REST simplifies integration and deployment while enabling layered, web-scale services. REST APIs in OpenAM implement REST in a way that reuses common HTTP verbs and decouples APIs from the programming languages that developers use to interact with them. OpenAM exposes REST APIs for many capabilities such as those in the following list.

- Authentication (including a callback mechanism so applications can work with all OpenAM authentication modules)
- Logout
- Managing groups
- Managing policy agent profiles
- Managing realms
- Managing user profiles

- OAuth 2.0 authorization
- OpenAM native authorization
- OpenID Connect 1.0 authorization
- Resetting forgotten passwords
- Token validation
- User self-registration



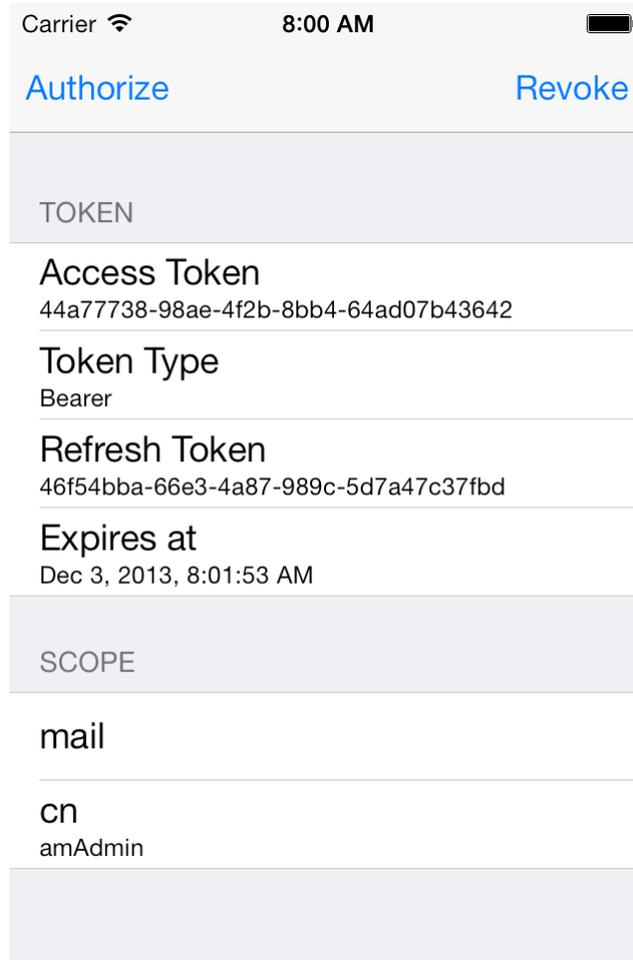
As OpenAM REST APIs in the *Developer's Guide* provide language-independent access, they make it easier to build cross-device applications. Developers can use the same APIs to access OpenAM both from web applications and also from native mobile applications.

Furthermore OpenAM REST APIs are built on an underlying common REST framework, designed to provide common access to resource providers. The common REST framework standardizes both how resource providers serve standard requests (create, read, update, delete, query, patch), and also how resource providers offer extended operations in a managed way (using actions). Applications built to interact with OpenAM REST APIs increasingly can interoperate with other products in the ForgeRock stack such as OpenIDM for identity management and OpenDJ for highly available data.

## 7.4. Getting Source Code for Sample Mobile Applications

You can get source code for sample mobile applications from ForgeRock's public source code repository under <https://svn.forgerock.org/commons/samples/mobile>.

For example, if you have a Mac running OS X 10.8 or later with Xcode installed, try the OpenAM OAuth 2.0 iOS Sample App.



## Chapter 8

# Configuring Password Reset

This chapter focuses on how to enable users to reset their own passwords in secure fashion.

## 8.1. About Password Reset

Users who know their passwords, but must reset them because for example the password is going to expire, can reset their passwords by successfully authenticating to OpenAM, visiting their end user pages, such as <http://openam.example.com:8080/openam/idm/EndUser>, and clicking Edit next to the Password field to display the change password page.

**Change Password for Kirsten Vaughan** OK Reset Close

\* Indicates required field

Type in your old password.

\* Old Password:

Type in the new password, then re-enter it.

\* New Password:

\* Re-Enter Password:

You therefore do not need to configure password reset for users who can remember their current password. Instead, you point them to the [idm/EndUser](#) page to let them do it themselves.

## 8.2. Resetting Forgotten Passwords

OpenAM can provide self-service password reset for forgotten passwords. To enable self-service password reset, you must configure the password reset service itself, which consists mainly of setting up secret questions, and configuring an SMTP mail server to send reset passwords to the users of the service.

**Tip**

Users must be able to access their mail after the service resets their passwords, or they will not be able to receive the new password. Do not therefore set up the service to reset the password used to access the email account specified in the user's profile.

### Procedure 8.1. To Set Up the Password Reset Service

You can configure the password reset service for OpenAM, letting each realm inherit the global settings. Alternatively, you can choose to configure the service only for an individual realm.

1. (Optional) When OpenAM is configured with default settings, it uses the `ldapService` authentication chain, which relies on the `DataStore` authentication module. The `DataStore` authentication module provides a generic authentication mechanism for OpenAM data stores, and therefore cannot handle specific data store settings, such as the directory server password policy setting to force password changes on reset. When you use settings the module cannot handle, then authentication can fail.

If you must configure the directory server to force password changes on reset, then also configure a separate authentication chain for users. The separate authentication chain must require the `LDAP` authentication module rather than the `DataStore` authentication module.

You can create and configure authentication chains, and assign them in the OpenAM console under Access Control > *Realm Name* > Authentication.

The OpenAM administrator, `amadmin`, uses the `DataStore` authentication module. If you set Access Control > /(Top Level Realm) > Authentication > Core > Organization Authentication Configuration to use your `LDAP` based authentication chain for users, let the Administrator Authentication Configuration continue to use the `DataStore` based authentication chain.

2. Configure the Password Reset service in one of the following ways.
  - To configure the service globally for all realms, login to OpenAM Console as administrator and browse to Configuration > Global > Password Reset in the Global Properties list.
  - To configure the service for a particular realm, login to OpenAM console as the realm administrator and browse to Access Control > *Realm Name* > Services, then click Add... to add a new Password Reset service configuration.
3. In the Password Reset page, use the following hints to adjust settings, and then save your work.

In addition to the User Validation and Secret Question values provided, you must configure at least the Bind DN and Bind Password of the user who can reset passwords in the LDAP data store.

#### User Validation

OpenAM uses this LDAP attribute and the value entered by the user to look up the user profile in the data store.

## Secret Question

This list corresponds to property values held in the file `amPasswordReset.properties` inside `openam-core-11.0.0.jar`, which you can find under `WEB-INF/lib/` where OpenAM is installed.

To make changes, extract a version from `openam-core-11.0.0.jar`, copy it to `WEB-INF/classes/` where OpenAM is deployed, and then edit `WEB-INF/classes/amPasswordReset.properties`.

Localized versions of this file are named `amPasswordReset_locale.properties`. You should localize only the questions at the end, leaving the rest of the localized file as is. For example if the default properties file contains:

```
favourite-restaurant=What is your favorite restaurant?
```

Then `WEB-INF/classes/amPasswordReset_fr.properties` ought to contain:

```
favourite-restaurant=Quel est votre restaurant préféré ?
```

After changing these files, you must restart OpenAM.

## Search Filter

An additional LDAP search filter you specify here is &-ed with the filter constructed for user validation to find the user entry in the data store.

## Base DN

If you specify no base DN for the search, the search for the user entry starts from the base DN for the realm.

## Bind DN

The DN of the user with access to change passwords in the LDAP data store.

## Bind Password

The password of the user with access to change passwords in the LDAP data store.

## Reset Password Creator

Classname of a plugin that implements the `PasswordGenerator` interface.

Default: `com.sun.identity.password.plugins.RandomPasswordGenerator`

## Password Reset Notification Class

Classname of a plugin that implements the `NotifyPassword` interface.

Default: `com.sun.identity.password.plugins.EmailPassword`

## Password Reset

Enables the service.

## Personal Question

When enabled, allows the user to create custom secret questions.

## Maximum Number of Questions

Maximum number of questions to ask during password reset.

## Force Change Password on Next Login

When enabled, the user must change her password next time she logs in after OpenAM resets her password.

## Password Reset Failure Lockout

When enabled, the user only gets the specified number of tries before her account is locked.

## Password Reset Failure Lockout Count

If Password Reset Failure Lockout is enabled, this specifies the maximum number of tries to reset a password within the specified interval before the user's account is locked.

## Password Reset Failure Lockout Interval

This interval applies when Password Reset Failure Lockout is enabled, and when Password Reset Failure Lockout Count is set. During this interval, a user can try to reset her password the specified number of times before being locked out. For example, if this interval is 5 minutes and the count is set to 3, a user gets 3 tries during a given 5 minute interval to reset her password.

## Email Address to Send Lockout Notification

This specifies the administrator address(es) which receive(s) notification on user account lockout. Each address must be a full email address such as `admin@example.com`, or `admin@host.domain`.

OpenAM must be able to send mail through an SMTP-capable service for this to work. See Procedure 8.2, "To Set Up SMTP Mail Notification".

## Warn User After N Failures

If you configure Password Reset Failure Lockout, set this to warn users who are about to use up their count of tries.

## Password Reset Failure Lockout Duration

If you configure Password Reset Failure Lockout, set this to a number of minutes other than 0 so that lockout is temporary, requiring only that the locked-out user wait to try again to reset her password, rather than necessarily require help from an administrator.

## Password Reset Lockout Attribute Name

If you configure Password Reset Failure Lockout, then OpenAM sets data store attribute to `inactive` upon lockout.

## Password Reset Lockout Attribute Value

If set to `inactive`, then a user who is locked out cannot attempt to reset her password if the Password Reset Failure Lockout Duration is 0.

## Password Reset E-mail Attribute Name

Identity attribute that holds the user's email address.

Default: `mail`

4. If you changed Secret Questions in the `WEB-INF/classes/amPasswordReset.properties` file or in any localized versions, restart OpenAM for the changes to take effect.

### Procedure 8.2. To Set Up SMTP Mail Notification

By default, OpenAM expects the SMTP service to listen on `localhost:25`. You can change these settings.

1. In the OpenAM console, click the Configuration > Servers and Sites > Default Server Settings.
2. In the Edit server-default page, scroll down to Mail Server to change the Mail Server Host Name or Mail Server Port Number.
3. Save your work.
4. By default, OpenAM sends password reset notifications from `<Password-Administrator>`.

To set a valid from address, extract `amPasswordResetModuleMsgs.properties` from `openam-core-11.0.0.jar`, copy it to `WEB-INF/classes/` where OpenAM is deployed, and then edit the file to change the `fromAddress.label` property value, as in the following example.

```
fromAddress.label=no-reply@example.com
```

Save your work, and then restart OpenAM for the properties file change to take effect.

### Procedure 8.3. To Prepare Users to Reset Passwords

Before a user can reset her password, she must choose answers for secret questions.

1. When her account is first created, direct the user to her `idm/EndUser` page, such as `http://openam.example.com:8080/openam/idm/EndUser`, where she can provide a valid email address to recover the reset password and can edit Password Reset Options.

**Password User**

\* Indicates required field

First Name:

\* Last Name:

\* Full Name:

Password: [Edit](#)

Email Address:

Telephone Number:

Home Address:

Password Reset Options: [Edit](#)

Universal ID: `id=pwduser,ou=user,dc=openam,dc=forgerock,dc=org`

By default OpenAM console redirects end users to this page when they login.

2. After the user updates her secret questions, she can use the password reset service when necessary.

**Password Reset Options**

Questions (2 Questions)

<input checked="" type="checkbox"/>	Question	Answer
<input checked="" type="checkbox"/>	What is your favorite restaurant?	<input type="text" value="Anything Tibetan"/>
<input checked="" type="checkbox"/>	What is your favorite password?	<input type="text" value="secret12"/>

#### Note

Answers to secret questions are case sensitive.

### Procedure 8.4. To Direct Users to Reset Passwords

Having setup her email and answers to secret questions, the user can use the reset password service. Create a test subject and use these steps to validate your configuration.

1. Send the user with a forgotten password to enter her user ID at the password reset URL.

If the user is in the default realm use `password` at the end of the URL to OpenAM, as in <http://openam.example.com:8080/openam/password>.

If the password reset service is enabled only for the user's realm and not the parent realm, or the realm to reset the password is different from the user's default realm, use `ui/PWResetUserValidation?realm=realm name`, as in <http://openam.example.com:8080/openam/ui/PWResetUserValidation?realm=realm name>.

# Password Reset User Validation

User ID:

NEXT

2. The user answers the specified questions, and clicks OK.

OpenAM resets the password, sending mail to the SMTP service you configured.

# Password Question for pwduser

What is your favorite restaurant?:

Anything Tibetan

What is your favorite password?:

secret12

OK

PREVIOUS

When the user clicks OK, OpenAM sends the email and shows a confirmation message.

The user receives the email with a line such as the following.

```
Your OpenAM password was changed to: 647bWlUw
```

3. The user logs in using the new password.

If you configured the system to force a change on password reset, then OpenAM requires the user to change her password.

## Chapter 9

# Configuring Single Sign-On within One Domain

This chapter describes the configuration of *Single Sign-On* (SSO) services for multiple resources on one domain. To understand how SSO works, you need to understand some key elements of the HTTP cookie, as described in RFC 6525, *HTTP State Management Mechanism*.

With SSO a user can access multiple independent services from a single session.

## 9.1. The Basics of the HTTP Cookie

Within an HTTP cookie, you can store a single custom *name=value* pair, such as *sessionId=value*. Other custom names within a cookie are as follows.

### Domain

Normally set to the domain from where the cookie was issued. To work with multiple subdomains, the `Domain` should be set to a URL like `Domain=example.net`. This is also known as the cookie domain, as defined in the *Configuration Reference* in the *Reference* chapter of the Reference document. A cookie domain set to `example.net` will work for subdomains such as `appl.example.net` and `service1.example.net`.

### Path

The directory in the URL to which the cookie applies. If the `Path=/openam`, the cookie applies to the `/openam` subdirectory of the FQDN, and lower level directories, including `openam/UI` and `openam/UI/Login`.

### Secure

If the `Secure` name is included, the cookie can be transferred only over HTTPS. When a request is made over HTTP, the cookie is not made available to the application.

### HttpOnly

When the `HttpOnly` name is included, that cookie will not be accessible through JavaScript. According to RFC 6265, the noted flag "instructs the user agent to omit the cookie when providing access to cookies via 'non-HTTP' APIs (such as a web browser API that exposes cookies to scripts)."

## Expires

The lifetime of a cookie can be limited, with an Expires name configured with a time, based on UTC (GMT).

### Note

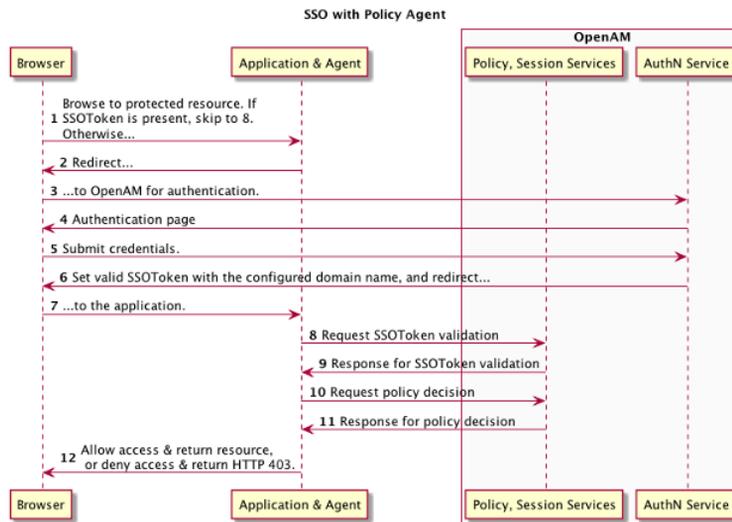
Be careful. Do not take a shortcut with a top-level domain. Web browser clients today are designed to ignore cookies set to top-level domains including `com`, `net`, and `co.uk`. In addition, a cookie with a value like `Domain=app1.example.net` will not work for similar subdomains such as `app2.example.net`.

## 9.2. Cookies and the SSO Session Process

OpenAM uses cookies to track user sessions. The diagram shown next illustrates how OpenAM assigns and tracks cookies.

In the diagram:

- The domain shown in the description is `example.net`
- The protected resource application can be found on `app.example.net`
- The OpenAM server is located on `sso.example.net`.



A client points his browser to a protected resource application. An agent on the application checks the client browser cookies for the presence of a session ID, a component of an SSO Token. If such a Session ID exists and is valid, the agent requests validation (see arrow 8).

If no valid session ID currently exists, the agent redirects the client to OpenAM for authentication (AuthN). The client is then sent to OpenAM for AuthN. If the client submits valid credentials, the AuthN service creates a session cookie. The SSO Token and configured domain name is embedded in that cookie. OpenAM issues an HTTP redirect to send the client browser back to the protected resource. The SSO Token is actually a Java Object.

The agent then verifies the validity of the session with the OpenAM session service, before granting access.

## 9.3. Potential Problems

In general, problems with SSO relate to some sort of mismatch of domain names. For example, a cookie that is configured on a third-level domain such as `sso.example.net` will not work with an application on a similar domain such as `app.example.net`. Even if the Session ID is valid, the application will not receive the SSO Token. The request is then redirected to OpenAM. The client gets what appears as a SSO Token in the diagram, which is actually a valid SSO tracking cookie that redirects immediately, and the cycle continues. Other issues that may lead to similar problems are shown here.

- When a cookie domain does not match a domain for the protected application

Assume the application is configured on a domain named `example.org`. That application will not receive an SSO Token configured on the `example.net` domain.

- When a third-level domain is used for the SSO Token

If an SSO Token is configured on `sso.example.net`, an application on `app.example.net` does not receive the corresponding cookie. In this case, the solution is to configure the SSO Token on `example.net`.

- When the Secure flag is used with a regular HTTP application

If you need encrypted communications for an application protected by OpenAM, use the Secure flag and make sure the application is accessible over HTTPS.

- When the path listed in the cookie does not match the path for the application

Perhaps the cookie is configured with an `/helloworld` path; that won't match an application that might be configured with an `/hellomars` path. In that case, the application will not receive the cookie.

- When an inappropriate name is used for the cookie domain

As noted earlier, client browsers are configured to ignore first-level domains such as `com` and `net` as well as functional equivalents such as `co.uk` and `co.jp`.

- When working with different browsers

The `name=value` pairs described earlier may not apply to all browsers. The requirements for an HTTP cookie sent to an IE browser may differ from the requirements for other standard browsers such as Firefox and Chrome. Based on anecdotal reports, IE does not recognize domain names that start

with a number. In addition, IE reportedly refuses cookies that include the underscore (\_) character in the FQDN.

## 9.4. Configure SSO on One Domain

Now that you have read about the SSO process, you should be able to set it up on a server configured with OpenAM and a web service protected by an OpenAM agent. The following procedure assumes that you know how to configure OpenAM, the Apache Web server, and associated OpenAM Apache agent.

### Procedure 9.1. Configure SSO on One Domain

1. Install OpenAM as described in the *OpenAM 11.0.0 Installation Guide*. This procedure uses a Server URL of <http://openam.example.net:8080/openam>.
2. Install the appropriate policy agent, as described in the *OpenAM Web Policy Agent 3.3.0 Installation Guide* or *OpenAM Java EE Policy Agent 3.3.0 Installation Guide*. This procedure uses an agent URL of <http://app.example.net:80>, and an agent name of `webagent1`.
3. Make sure that both URLs are configured with IP addresses, as described in the chapter on *Installing OpenAM Core Services* in the *Installation Guide*.
4. Return to the OpenAM server on <http://openam.example.net:8080/openam>. Log in as the administrative user, normally `amadmin`. To activate and configure the agent, follow the procedure described in the *OpenAM Web Policy Agent 3.3.0 Installation Guide* or *OpenAM Java EE Policy Agent 3.3.0 Installation Guide*.
5. Now you can configure SSO Only mode. In the OpenAM console, click `Access Control > Realm Name > Agents > webagent1`. Scroll down to SSO Only Mode and activate the Enabled box.
6. Save your changes.
7. Make sure you have configured the SSO domain, in this case, `example.net`. Click `Configuration > System > Platform`. Make sure `example.net` (or your chosen domain) is selected as a cookie domain.
8. Save your changes.
9. Restart the web server. The agent should be active. You should now be able to log out of the OpenAM server.
10. Verify the agent URL, in this case, <http://app.example.net>. The OpenAM web agent should now redirect requests to the OpenAM server.

If you want to configure OpenAM and an application on two different cookie domains, such as `example.org` and `example.net`, you will need to set up Cross-Domain SSO (CDSSO). For more information, see the chapter on *Configuring Cross-Domain Single Sign On*.

## Chapter 10

# Configuring Cross-Domain Single Sign On

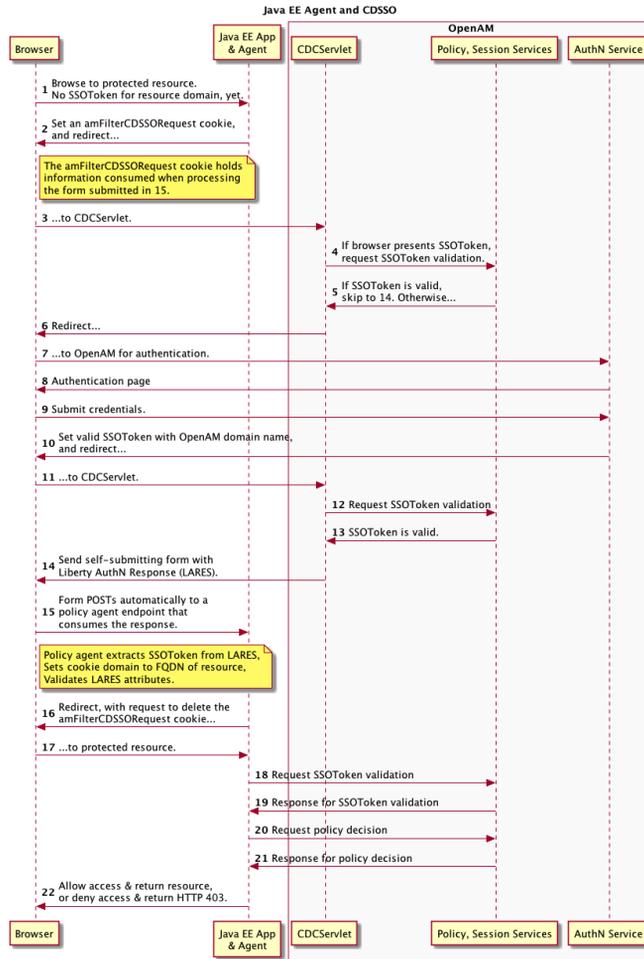
This chapter shows you how to configure cross-domain single sign on (CDSSO). When you have multiple domains in a single organization, CDSSO lets your OpenAM servers in one domain work with policy agents from other domains.

CDSSO is an OpenAM-specific capability. For single sign on across multiple organizations or when integrating with other access management software, use OpenAM's federation capabilities.

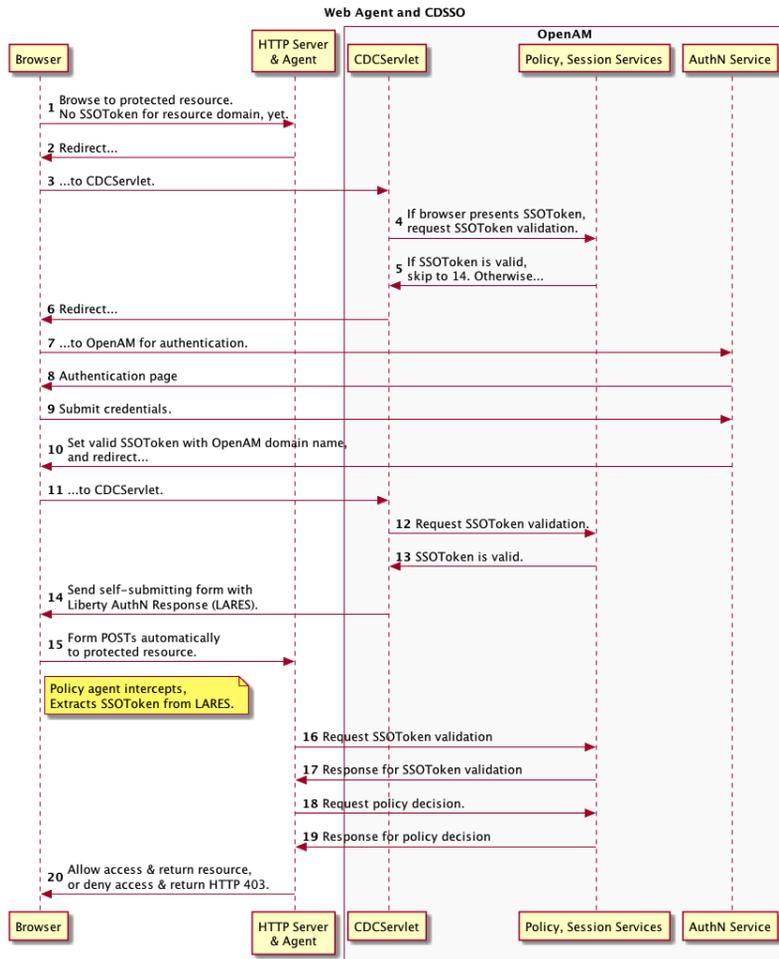
*Cross-domain single sign on* provides a safe mechanism for managing access across multiple different domains that you control. CDSSO lets OpenAM authenticate users redirected by policy agents in other DNS domains.

Single sign on depends on cookies to store session information. Yet for security reasons, browsers do not let a web site in one domain to get access to a cookie from another domain. With CDSSO, the policy agents work around this by negotiating with OpenAM to allow access.

The Java EE policy agent allows CDSSO by using a mechanism to write the SSO token from OpenAM authentication to a cookie with the domain the host where the agent runs. The following sequence diagram illustrates this mechanism.



Whereas the Java EE policy agent has an endpoint specifically to handle the cookie domain translation, the web policy agent handles the request directly as shown in the following sequence diagram.



This chapter includes the following procedures.

- Procedure 10.1, "To Enable CDSSO For a Java EE Policy Agent"
- Procedure 10.2, "To Enable CDSSO For a Web Policy Agent"
- Procedure 10.3, "To Indicate Progress During CDSSO Login"
- Procedure 10.4, "To Protect Against CDSSO Cookie Hijacking"
- Procedure 10.5, "Using SAML 2.0 With Policy Agents"

### Procedure 10.1. To Enable CDSSO For a Java EE Policy Agent

1. In the OpenAM console, browse to Access Control > *Realm Name* > Agents > J2EE > *Agent Name* > SSO.
2. Scroll down and enable Cross Domain SSO.
3. Check that the CDSSO Redirect URI is set.

Depending on where you deployed your Java EE agent application, the default is something like [/agentapp/sunwCDSSORedirectURI](#).

4. Set the list of URLs for CDSSO Servlet URL to the Cross Domain Controller Servlet URLs of the servers the agent accesses, such as [http://openam.example.com:8080/openam/cdcservlet](#).

If the agent accesses OpenAM through a load balancer, use the load balancer URLs, such as [http://load-balancer.example.com:8080/openam/cdcservlet](#).

5. Leave the CDSSO Clock Skew set to 0.

Make sure instead that the clocks on the servers where you run OpenAM and policy agents are synchronized.

6. Set the list of URLs for CDSSO Trusted ID Provider to the Cross Domain Controller Servlet URLs of the OpenAM servers the agent accesses, such [http://openam.example.com:8080/openam/cdcservlet](#).

This list should include one CDC Servlet URL for every OpenAM server the agent might access. You do not need to include site or load balancer URLs.

7. (Optional) To protect the SSO token from network snooping, you can select CDSSO Secure Enable to mark the SSO token cookie as secure.

If you select this, then the SSO token cookie can only be sent over a secure connection (HTTPS).

8. Add the domains involved in CDSSO in the CDSSO Domain List.

9. If necessary, update the Agent Root URL for CDSSO list on the Global tab page.

If the policy agent is on a server with virtual host names, add the virtual host URLs to the list.

If the policy agent is behind a load balancer, add the load balancer URL to the list.

10. Save your work.

### Procedure 10.2. To Enable CDSSO For a Web Policy Agent

1. In the OpenAM console, browse to Access Control > *Realm Name* > Agents > Web > *Agent Name* > SSO.

2. Enable Cross Domain SSO.
3. Set the list of URLs for CDSSO Servlet URL to the Cross Domain Controller Servlet URLs of the servers the agent accesses, such as `http://openam.example.com:8080/openam/cdcservlet`.

If the agent accesses OpenAM through a load balancer, use the load balancer URLs, such as `http://load-balancer.example.com:8080/openam/cdcservlet`.

4. Add the domains involved in CDSSO in the Cookies Domain List.
5. If necessary, update the Agent Root URL for CDSSO list on the Global tab page.  
If the policy agent is on a server with virtual host names, add the virtual host URLs to the list.  
If the policy agent is behind a load balancer, add the load balancer URL to the list.
6. Save your work.

### Procedure 10.3. To Indicate Progress During CDSSO Login

The default self-submitting form page that OpenAM presents to users contains hidden fields, but is otherwise blank. If you want to show users that the operation is in progress, then customize the necessary JSP.

1. Edit a copy of the file `config/federation/default/cdclogin.jsp` to add a clue that SSO is in progress, such as an image.

You can find this file where you deployed OpenAM, such as `/path/to/tomcat/webapps/openam/config/federation/default/cdclogin.jsp`.

When you add an image or other presentation element, make sure that you retain the form and JavaScript as is.

2. Unpack OpenAM-11.0.0.war, and replace the file with your modified version.  
Also include any images you reference in the page.
3. Pack up your custom version of OpenAM, and then deploy it in your web container.

### Procedure 10.4. To Protect Against CDSSO Cookie Hijacking

When cookies are set for an entire domain such as `.example.com`, an attacker who steals a cookie can use it from any host in the domain such as `untrusted.example.com`. Cookie hijacking protection restricts cookies to the fully-qualified domain name (FQDN) of the host where they are issued, such as `openam-server.example.com` and `server-with-agent.example.com`, using CDSSO to handle authentication and authorization.

For CDSSO with cookie hijacking protection, when a client successfully authenticates OpenAM issues the master SSO token cookie for its FQDN. OpenAM issues *restricted token* cookies for the other

FQDNs where the policy agents reside. The client ends up with cookies having different session identifiers for different FQDNs, and the OpenAM server stores the correlation between the master SSO token and restricted tokens, such that the client only has one master session internally in OpenAM.

To protect against cookie hijacking you restrict the OpenAM server domain to the server where OpenAM runs. This sets the domain of the SSO token cookie to the host running the OpenAM server that issued the token. You also enable use of a unique SSO token cookie. For your Java EE policy agents, you enable use of the unique SSO token cookie in the agent configuration as well.

1. In the OpenAM console, browse to Configuration > System > Platform.
2. Remove the domain such as `.example.com` from the Cookies Domains list, and replace it with the server host name such as `openam.example.com`, or if OpenAM is behind a load balancer with the load balancer host name, such as `load-balancer.example.com`.
3. Save your work.
4. In the OpenAM console, browse to Configuration > Servers and Sites > Default Server Settings > Advanced, and then make the necessary changes.
  - a. Change the setting for the property `com.sun.identity.enableSSOTokenCookie` to `true`, from the default `false`.
  - b. Make sure that the property `com.sun.identity.authentication.uniqueCookieName` is set to the name of the cookie that will hold the URL to the OpenAM server that authenticated the user.

The default name is `sunIdentityServerAuthNServer`.

Save your work.

5. Browse to Configuration > Servers and Sites > *Server Name* > Advanced, and add the property `com.sun.identity.authentication.uniqueCookieDomain`, setting the value to the fully-qualified domain name of the current OpenAM server, such as `openam.example.com`.

Then Save your work.

6. (Optional) For each Java EE policy agent, browse in the OpenAM console to Access Control > *Realm Name* > Agents > J2EE > *Agent Name* > Advanced > Custom Properties, and add `com.sun.identity.enableSSOTokenCookie=true` to the list.
7. Save your work.
8. Restart OpenAM or the container in which it runs for the configuration changes to take effect.

### Procedure 10.5. Using SAML 2.0 With Policy Agents

The federation mechanism associated with SAML can be used as an alternative to CDSSO for both Web and Java EE policy agents. This assumes OpenAM is configured as an IDP on one domain, and

the desired policy agent protects resources on behalf of a second instance of OpenAM, where that instance acts as an SP on a second domain.

While using SAML 2.0 adds complexity, it supports attribute mapping, which may be useful when the two domains are associated with datastores with different attributes.

1. Install a policy agent. The basic process for installing policy agents is available in the *OpenAM Web Policy Agent Installation Guide* and the *OpenAM Java EE Policy Agent Installation Guide*.
2. Now you can replace the given OpenAM Login URL and OpenAM Logout URLs with SAML2 URLs configured in the section on *Managing SAML 2.0 Federation*.
3.
  - If you have configured the Web policy agents to store their properties centralized on an OpenAM server, navigate to the URL for the OpenAM console. Select Access Control > *Realm Name* > Agents > Web > *Agent Name* > OpenAM Services.

For the Web Agent, under the OpenAM Services tab, in the Agent Logout URL section, set up a list of application logout URLs. In the Logout Redirect URL text box, enter an appropriate URL to redirect the user after logout.

- Alternatively, if the Web policy agents are set up to store properties on local systems, find the `OpenSSOAgentConfiguration.properties` file in the `/path/to/agent/config/` directory. You can specify OpenAM Login and Logout URLs with the `com.sun.identity.agents.config.login.url` and `com.sun.identity.agents.config.logout.url` attributes, respectively.

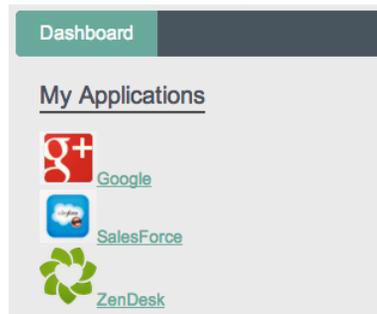
## Chapter 11

# Configuring the Dashboard Service

This chapter shows how to configure the OpenAM *Dashboard*.

## 11.1. About the Dashboard Service

The Dashboard service provides the end user with an interface to access applications secured by OpenAM, both cloud-based applications like SalesForce and internal applications protected by policy agents. The Dashboard service uses SSO to login to the applications when the user clicks on the application icon. For some apps, like SalesForce, you will want to limit access to only a few users. Other apps, like Google Mail or Drive, you will probably want to make available to all users.



The Dashboard service is meant to give users a single place to access their applications. Keep in mind that this does not limit user access, only what appears on the user Dashboard.

There are three stages to setting up the Dashboard service.

- Setup the Dashboard service and add applications.
- Add the service to the realms.
- Assign users applications so that they appear on the users' Dashboards. This can be done manually or through a provisioning solution.

## 11.2. Setting Up the Dashboard Service

Making some applications universally available ensures that all users have the same basic applications. However, some of your applications should be protected from the majority of your users. You will need to single out which users will include the application on their Dashboard.

There are three default applications in the Dashboard service: Google, Salesforce, and ZenDesk.

### *Procedure 11.1. To Add Applications from the Dashboard*

You can add applications to the Dashboard service with the following steps. All fields except the Dashboard Class name and ICF Identifier are required for the application to work properly from the Dashboard.

1. Login to the OpenAM console as OpenAM Administrator, `amadmin`.
2. On the Configuration tab > Global > Dashboard click New to add a new application to the Dashboard service and to provide the information needed to connect to the app.
3. Provide a unique name for the application.
4. Add a Dashboard Class Name that identifies how the end user will access the app, such as `SAML2ApplicationClass` for a SAML 2.0 application.
5. Add a Dashboard Name for the application.
6. Add a Dashboard Display Name. This name is what the end user will see, such as Google.
7. Add the Dashboard Icon you would like the end user to see for the application. Either use a fully-qualified URL or an appropriate relative URL so that the icon is rendered properly on the user Dashboard.
8. Add the Dashboard Login URL to point to the location the end user will go to once they click on the icon.
9. Leave the ICF Identifier blank.
10. Click Add when you are done.

## 11.3. Configuring Dashboard Service for a Realm

### *Procedure 11.2. To Add the Application Dashboard Service to a Realm*

You must add the Dashboard service to a realm before it will be available. The following instructions show you how to add an application to a single realm. Before you begin, make sure you have the name of the application as it appears on the Secondary Configuration Instance table under Configuration > Global > Dashboard.

1. On the Access Control > *Realm Name* > Services, click Add....
2. Select the Dashboard service, then click Next.
3. Add or remove the applications you would like to appear on the Dashboard service for the realm.
4. Click Finish when you are done.

## 11.4. Adding Applications to a User's Dashboard

### *Procedure 11.3. To Add an Application to a User's Dashboard*

Use the following steps to add an application to a user's Dashboard.

1. On the Access Control > *Realm Name* > Subjects, click the user identifier to edit the user's profile.
2. Under Services, click Dashboard.
3. Add the application beside the user name under the user's Assigned Dashboard list.
4. Click Save.

### *Procedure 11.4. Removing User Access to an Application*

You may need to remove an application from user's Dashboard, but you don't want to entirely delete the user. The following steps walk you through removing an application from a user's Dashboard.

1. On the Access Control > *Realm Name* > Subjects, click the user identifier to edit the user's profile.
2. Under Services, click Dashboard.
3. Delete the application beside the user name under the user's Assigned Dashboard list.
4. Click Save.

## Chapter 12

# Managing SAML 2.0 Federation

This chapter addresses how to set up and manage SAML 2.0 SSO for single sign on and single log out across resources belonging to organizations participating in a circle of trust.

## 12.1. About SAML 2.0 SSO & Federation

SAML 2.0 SSO is part of federated access management. Federation lets access management cross organizational boundaries. Federation helps organizations share identities and services without giving away their identity information, or the services they provide.

To bridge heterogeneous systems, federation requires interoperability, and thus depends on standards for orchestrating interaction and exchanging information between providers. OpenAM federation relies on standards such as Security Assertion Markup Language (SAML) 2.0. SAML 2.0 describes the messages, how they are relayed, how they are exchanged, and common use cases.

To achieve SAML 2.0 SSO, OpenAM separates *identity providers* from *service providers*, lets you include them in a *circle of trust*, and has you configure how the providers in the circle of trust interact.

- An identity provider stores and serves identity profiles, and handles authentication.
- A service provider offers services that access protected resources, and handles authorization.
- A circle of trust groups at least one identity provider and at least one service provider who agree to share authentication information, with assertions about authenticated users that let service providers make authorization decisions.

Providers in a circle of trust share *metadata*, configuration information that federation partners require to access each others' services.

- SAML 2.0 SSO maps attributes from accounts at the identity provider to attributes on accounts at the service provider. The identity provider makes assertions to the service provider, for example to attest that a user has authenticated with the identity provider. The service provider then consumes assertions from the identity provider to make authorization decisions, for example to let an authenticated user complete a purchase that gets charged to the user's account at the identity provider.

In federation deployments where not all providers support SAML 2.0, OpenAM can act as a multi-protocol hub, translating for providers who rely on other and older standards such as SAML 1.x,

Liberty Alliance Project frameworks, and WS-Federation (for integration with Active Directory Federation Services, for example).

## 12.2. Setting Up SAML 2.0 SSO

Before you set up SAML 2.0 SSO in OpenAM, you must:

- Know which providers participate in the circle of trust.
- Know how OpenAM installations act as identity providers, or service providers.
- Agree with other providers on a synchronized time service.
- For identity information exchanged with other participants in a circle of trust, define how to map shared user attributes. Local user profile attribute names should map to user profile attribute names at other providers.

For example, if you exchange user identifiers with your partners, and you call it `uid` whereas another partner calls it `userid`, then you map your `uid` to your partner's `userid`.

- Import the keys used to sign assertions into the JKS key store in your OpenAM configuration directory. You can use the Java `keytool` command.

The OpenAM configuration key store is located at the top level of the configuration directory, such as `$HOME/openam/keystore.jks`. The password, stored in `$HOME/openam/.keypass`, is `changeit` by default. Also by default the only key available is for a self-signed certificate (alias: `test`) installed with OpenAM.

During set up, you must share metadata for providers that you host with other providers in the circle of trust. You must also configure remote providers, connecting to other providers by importing their metadata.

In OpenAM terms, a hosted provider is one served by the current OpenAM server, whereas a remote provider is one hosted elsewhere.

This section covers the following topics.

- Procedure 12.1, "To Create a Hosted Identity Provider"
- Procedure 12.2, "To Create a Hosted Service Provider"
- Procedure 12.3, "To Create a Remote Identity Provider"
- Procedure 12.4, "To Create a Remote Service Provider"
- Procedure 12.5, "To Create a Fedlet for Service Providers"
- Section 12.2.1, "Deploying the Identity Provider Discovery Service"

### *Procedure 12.1. To Create a Hosted Identity Provider*

1. On the OpenAM console Common Tasks page, click Create Hosted Identity Provider.
2. Unless you already have metadata for the provider, accept the Name for this identity provider in the field provided, or provide your own unique identifier.

The default name is the URL to the current server which hosts the identity provider.

3. Select the Signing Key you imported into the OpenAM key store.
4. Either add the provider to the circle of trust you already created, or select Add to new and provide a New Circle of Trust name.
5. For the attributes you share, map service provider attribute names (Name in Assertion), to user profile names from your identity repository (Local Attribute Name).

The default IDP mapping implementation allows you to add static values in addition to values taken from the user profile. You add a static value by enclosing the profile attribute name in double quotes ("), as in the following examples.

To add a static SAML attribute called `partnerID` with a value of `staticPartnerIDValue`, add `partnerID` as the Name in Assertion with `"staticPartnerIDValue"` as the Local Attribute Name.

6. Click Configure to save your configuration.
7. Export the XML-based metadata from your provider to share with other providers in your circle of trust.

```
$ curl -o metadata.xml
http://www.idp.example:8080/openam/saml2/jsp/exportmetadata.jsp?entityid=
http://www.idp.example:8080/openam&realm=/realm-name
```

When you have configured only the top-level realm, `/`, you can omit the query string.

Alternatively, provide the URL, to other providers so they can load the metadata.

### *Procedure 12.2. To Create a Hosted Service Provider*

1. On the OpenAM console Common Tasks page, click Create Hosted Service Provider.
2. Unless you already have metadata for the provider, accept the Name for this service provider in the field provided, or provide your own unique identifier.

The default name is the URL to the current server which hosts the service provider.

3. Either add the provider to the circle of trust you already created, or select Add to new and provide a New Circle of Trust name.
4. (Optional) If the identity provider has not already mapped the attributes you share, map identity provider attribute names (Name in Assertion), to user profile names from your identity repository (Local Attribute Name).

5. Click Configure to save your configuration.
6. Export the XML-based metadata from your provider to share with other providers in your circle of trust.

```
$ curl -o metadata.xml
http://www.sp.example:8080/openam/saml2/jsp/exportmetadata.jsp?entityid=
http://www.sp.example:8080/openam&realm=/realm-name
```

When you have configured only the top-level realm, */*, you can omit the query string.

Alternatively, provide the URL, to other providers so they can load the metadata.

### *Procedure 12.3. To Create a Remote Identity Provider*

1. Obtain the identity provider metadata, or the URL where you can obtain it.
2. On the OpenAM console Common Tasks page, click Register Remote Identity Provider.
3. Provide the identity provider metadata or link to obtain metadata.
4. Either add the provider to the circle of trust you already created, or select Add to new and provide a New Circle of Trust name.
5. Click Configure to save your configuration.

### *Procedure 12.4. To Create a Remote Service Provider*

1. Obtain the service provider metadata, or the URL where you can obtain it.
2. On the OpenAM console Common Tasks page, click Register Remote Service Provider.
3. Provide the identity provider metadata or link to obtain metadata.
4. (Optional) If the identity provider has not already mapped the attributes you share, map identity provider attribute names (Name in Assertion), to user profile names from your identity repository (Local Attribute Name).
5. Either add the provider to the circle of trust you already created, or select Add to new and provide a New Circle of Trust name.
6. Click Configure to save your configuration.

### *Procedure 12.5. To Create a Fedlet for Service Providers*

When your organization acts as the identity provider, and you want quickly to enable service providers to federate their services with yours, you can provide them with a *fedlet*. A fedlet is a small Java or .NET web application that can act as a service provider for a specific identity provider without requiring that you install all of OpenAM.

Fedlets support the following SAML 2.0 features.

*Table 12.1. Fedlet Support for SAML 2.0 Features*

SAML 2.0 Feature	Java Fedlet	.NET Fedlet
IDP & SP-initiated Single Sign-On (HTTP Artifact)	Supported	Supported
IDP & SP-initiated Single Sign-On (HTTP POST)	Supported	Supported
IDP & SP-initiated Single Logout (HTTP POST)	Supported	Supported
IDP & SP-initiated Single Logout (HTTP Redirect)	Supported	Supported
Sign Requests & Responses	Supported	Supported
Encrypt Assertion, Attribute, & NameID Elements	Supported	Supported
Export SP Metadata	Supported	Supported
Attribute Queries	Supported	Supported
XACML Requests	Supported	Not supported
Multiple IDPs	Supported	Supported
External IDP Discovery Service	Supported	Supported
Bundled IDP Reader Service for Discovery	Supported	Not supported

For more information on using fedlets, see *Using Fedlets in Java Web Applications* in the *Developer's Guide* and *Using Fedlets in .NET Applications* in the *Developer's Guide* in the *Developer's Guide*.

The following procedure describes how to create a Java Fedlet.

1. (Optional) If you have not done so already, set up your identity provider.
2. Enter the URL where the service provider will deploy the fedlet you create, and name the fedlet. If you create multiple fedlets, use the URL as a unique name that shows who has deployed the fedlet.
3. For the attributes you share, map service provider attribute names (Name in Assertion), to user profile names from your identity repository (Local Attribute Name).
4. Click Create to generate the `Fedlet.zip` file under the OpenAM configuration directory, such as `$HOME/openam/myfedlets/httpwwwexamplecom80myapp/Fedlet.zip`.
5. Give the `Fedlet.zip` file to the service provider for deployment.

### 12.2.1. Deploying the Identity Provider Discovery Service

When your circle of trust includes multiple identity providers, then service providers must discover which identity provider corresponds to a request. You can deploy the identity provider discovery service for this purpose as a separate web application.

Browsers only send cookies for the originating domain. Therefore when a browser accesses the service provider in the `www.sp.example` domain, the service provider has no way of knowing whether the user has perhaps already authenticated at `www.this-idp.example` or at `www.that-idp.example`. The providers therefore host an identity provider discovery service in a common domain, such as `www.disco.example`, and use that service to discover where the user logged in. The identity provider discover service essentially writes and reads cookies from the common domain. The providers configure their circle of trust to use the identity provider discovery service as part of SAML 2.0 federation.

Deploying the identity provider discovery service involves the following stages.

1. Deploy the `.war` file into your web application container.
2. Configure the discovery service.
3. Add the identity provider discovery service endpoints for writing cookies to and reading cookies from the common domain to the circle of trust configurations for the providers.
4. Share metadata between identity providers and the service provider.

### *Procedure 12.6. To Deploy the Discovery Service on Tomcat*

How you deploy the discovery service `.war` file depends on your web application container. The procedure in this section shows how to deploy on Apache Tomcat.

1. Copy the `IDPDiscovery-11.0.0.war` file to the `webapps/` directory.

```
$ cp ~/Downloads/openam/IDPDiscovery-11.0.0.war /path/to/tomcat/webapps/disco.war
```

2. Access the configuration screen through your browser.

In this example, Apache Tomcat listens for HTTP requests on `www.disco.example:8080`, and Tomcat has unpacked the application under `/disco`, so the URL is `http://www.disco.example:8080/disco`, which redirects to `Configurator.jsp`.

### *Procedure 12.7. To Configure the Discovery Service*

1. Configure the identity provider discovery service.

## Configuring IDP Discovery Service

Please provide the IDP Discovery service information

Debug directory:	<input type="text" value="/tmp/debug"/>
Debug Level:	<input type="text" value="error"/>
Cookie Type:	<input checked="" type="radio"/> PERSISTENT <input type="radio"/> SESSION
Cookie Domain:	<input type="text" value=".disco.example"/>
Secure Cookie:	<input type="radio"/> True <input checked="" type="radio"/> False
Encode Cookie:	<input checked="" type="radio"/> True <input type="radio"/> False
HTTP-Only Cookie:	<input type="radio"/> True <input checked="" type="radio"/> False
	<input type="button" value="Configure"/> <input type="button" value="Reset"/>

Hints for discovery service configuration parameters follow.

### Debug Directory

The discovery service logs to flat files in this directory.

### Debug Level

Default is `error`. Other options include `error`, `warning`, `message`, and `off`.

Set this to `message` in order to see the service working when you run your initial tests.

### Cookie Type

Set to PERSISTENT if you have configured OpenAM to use persistent cookies, meaning single sign on cookies that can continue to be valid after the browser is closed.

### Cookie Domain

The cookie domain is the common cookie domain used in your circle of trust for identity provider discovery, in this case `.disco.example`.

### Secure Cookie

Set this to true if clients should only return cookies when a secure connection is used.

### Encode Cookie

Leave this true unless your OpenAM installation requires that you do not encode cookies. Normally cookies are encoded such that cookies remain valid in HTTP.

## HTTP-Only Cookie

Set to true to use HTTPOnly cookies if needed to help prevent third-party programs and scripts from accessing the cookies.

2. Restrict permissions to the discovery service configuration file in `$HOME/libIDPDiscoveryConfig.properties`, where `$HOME` corresponds to the user who runs the web container where you deployed the service.

### *Procedure 12.8. To Add the Discovery Service to Your Circles of Trust*

Each provider has a circle of trust including itself. You configure each of these circles of trust to use the identity provider discovery service as described in the following steps.

1. On the service provider console, login as OpenAM Administrator.
2. On the service provider console, under Federation > Circle of Trust > *Circle of Trust Name* add SAML2 Writer and Reader Service URLs for the identity provider discovery service endpoints, and Save your work.

In this example, the writer URL is `http://www.disco.example:8080/disco/saml2writer`, and the reader URL is `http://www.disco.example:8080/disco/saml2reader`.

3. On each identity provider console, login as OpenAM Administrator.
4. On the identity provider console, under Federation > Circle of Trust Configuration > *Circle of Trust Name* also add SAML2 Writer and Reader Service URLs for the identity provider discovery service endpoints, and Save your work.

### *Procedure 12.9. To Share Identity & Service Provider Metadata*

Before performing these steps, install the administration tools for each provider as described in *To Set Up Administration Tools* in the *Installation Guide*. The administration tools include the **ssoadm** tool that you need to export metadata.

1. On each identity provider console, register the service provider as a remote service provider adding to the circle of trust you configured to use the identity provider discovery service.

The URL to the service provider metadata is something like `http://www.sp.example:8080/openam/saml2/jsp/exportmetadata.jsp`.

2. Obtain metadata for each identity provider.

```

$ ssh www.this-idp.example
$ cd /path/to/openam-tools/admin/openam/bin
$ ./ssoadm create-metadata-templ -y "http://www.this-idp.example:8080/openam"
-u amadmin -f /tmp/pwd.txt -i /idp -m this-standard.xml -x this-extended.xml
Hosted entity configuration was written to this-extended.xml.
Hosted entity descriptor was written to this-standard.xml.

$ ssh www.that-idp.example
$ cd /path/to/openam-tools/admin/openam/bin
$ ./ssoadm create-metadata-templ -y "http://www.that-idp.example:8080/openam"
-u amadmin -f /tmp/pwd.txt -i /idp -m that-standard.xml -x that-extended.xml

Hosted entity configuration was written to that-extended.xml.
Hosted entity descriptor was written to that-standard.xml.

```

- For each identity provider extended metadata file, change the value of the `hosted` attribute to `0`, meaning the identity provider is remote.
- On the service provider, add the identity providers to the circle of trust using the identity provider metadata.

```

$ ssh www.sp.example
$ cd /path/to/openam-tools/admin/openam/bin
$ ./ssoadm import-entity -t discocot -m ~/Downloads/this-standard.xml
-x ~/Downloads/this-extended.xml -u amadmin -f /tmp/pwd.txt

Import file, /Users/mark/Downloads/this-standard.xml.
Import file, /Users/mark/Downloads/this-extended.xml.
$ ./ssoadm import-entity -t discocot -m ~/Downloads/that-standard.xml
-x ~/Downloads/that-extended.xml -u amadmin -f /tmp/pwd.txt

Import file, /Users/mark/Downloads/that-standard.xml.
Import file, /Users/mark/Downloads/that-extended.xml.

```

- Test your work by using the Federation Connectivity Test that you start from the service provider console under Common Tasks > Test Federation Connectivity.

When the test is done, you can see messages from the `CookieWriterServlet` in the `libIDPDiscovery` log file where you set up logging when you configured the identity provider discovery service, such as `/tmp/debug/libIDPDiscovery`. Output generated during a test follows, with some lines folded to fit on the printed page.

```

08/08/2012 11:43:38:341 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieUtils.init : idpDiscoveryOnlyWar=true
08/08/2012 11:43:38:341 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieWriterServlet Initializing...
08/08/2012 11:43:38:341 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieWriterServlet.doGetPost: Preferred Cookie Name is _saml_idp
08/08/2012 11:43:38:341 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieWriterServlet.doGetPost: URL Scheme is null, set to https.
08/08/2012 11:43:38:341 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieWriterServlet.doGetPost: Preferred IDP Cookie Not found
08/08/2012 11:43:38:342 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieWriterServlet.doGetPost: Cookie Type is PERSISTENT
08/08/2012 11:43:38:342 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieWriterServlet.doGetPost: Cookie value is

```

```
aHR0cDovL3d3dy50aGF0LWlkCC5jb2060DA4MC9vcGVuYW0=  
08/08/2012 11:43:38:342 AM CEST: Thread[http-bio-8080-exec-4,5,main]  
CookieWriterServlet.doGetPost: Preferred Cookie Name _saml_idp  
08/08/2012 11:43:38:343 AM CEST: Thread[http-bio-8080-exec-4,5,main]  
CookieWriterServlet.doGetPost: Redirect to  
http://www.that-idp.example:8080/openam/SSORedirect/metaAlias/idp?resInfoID=  
s28bc4db004f1365d78d07d69846c54a3c850fe801  
08/08/2012 11:43:46:957 AM CEST: Thread[http-bio-8080-exec-4,5,main]  
CookieWriterServlet.doGetPost: Preferred Cookie Name is _saml_idp  
08/08/2012 11:43:46:957 AM CEST: Thread[http-bio-8080-exec-4,5,main]  
CookieUtils.cookieValue=aHR0cDovL3d3dy50aGF0LWlkCC5jb2060DA4MC9vcGVuYW0=  
result=aHR0cDovL3d3dy50aGF0LWlkCC5jb2060DA4MC9vcGVuYW0=  
08/08/2012 11:43:46:957 AM CEST: Thread[http-bio-8080-exec-4,5,main]  
CookieWriterServlet.doGetPost: Cookie Type is PERSISTENT  
08/08/2012 11:43:46:957 AM CEST: Thread[http-bio-8080-exec-4,5,main]  
CookieWriterServlet.doGetPost: Cookie value is  
aHR0cDovL3d3dy50aGF0LWlkCC5jb2060DA4MC9vcGVuYW0=  
08/08/2012 11:43:46:957 AM CEST: Thread[http-bio-8080-exec-4,5,main]  
CookieWriterServlet.doGetPost: Preferred Cookie Name _saml_idp  
08/08/2012 11:43:46:957 AM CEST: Thread[http-bio-8080-exec-4,5,main]  
CookieWriterServlet.doGetPost: Redirect to  
http://www.that-idp.example:8080/openam/SSORedirect/metaAlias/idp?resInfoID=  
s2ce9c465cf39c96f31e1dcf009cf9943695d82901
```

## 12.3. Configuring Identity Providers

Once you have set up an identity provider, you can configure it through the OpenAM console under Federation > Entity Providers > *Provider Name*.

### 12.3.1. Hints for Assertion Content

Use the following hints to adjust settings on the Assertion Content tab page.

#### *Signing and Encryption*

##### **Request/Response Signing**

Specifies what parts of messages the identity provider requires the service provider to sign digitally.

##### **Encryption**

When selected, the service provider must encrypt NameID elements.

##### **Certificate Aliases**

Specifies aliases for certificates in the OpenAM key store that are used to handle digital signatures, and to handle encrypted messages.

Specify a Key Pass if the private key password is different from the key store password, which is stored encrypted in the `.keypass` file for the server. For instructions on working with key pairs, also see *To Change the Signing Key for Federation*.

## *NameID Format*

### **NameID Format List**

Specifies the supported name identifiers for users that are shared between providers for single sign on. If no name identifier is specified when initiating single sign on, then the identity provider uses the first one in the list.

### **NameID Value List**

Maps name identifier formats to user profile attributes. The `persistent` and `transient` name identifiers need not be mapped.

## *Authentication Context*

### **Mapper**

Specifies a class that implements the `IDPAuthnContextMapper` interface and sets up the authentication context.

### **Default Authentication Context**

Specifies the authentication context used if no authentication context specified in the request.

### **Supported Contexts**

Specifies the supported authentication contexts, where the Key and Value can specify a corresponding OpenAM authentication method, and the Level corresponds to an authentication module authentication level.

## *Assertion Time*

### **Not-Before Time Skew**

Grace period in seconds for the `NotBefore` time in assertions.

### **Effective Time**

Validity in seconds of an assertion.

## *Basic Authentication*

### **Enabled, User Name, Password**

When enabled, authenticate with the specified user name and password at SOAP end points.

## Assertion Cache

### Enabled

When enabled, cache assertions.

## 12.3.2. Hints for Assertion Processing

Use the following hints to adjust settings on the Assertion Processing tab page.

## Attribute Mapper

### Attribute Mapper

Specifies a class that implements the attribute mapping.

The default implementation attempts to retrieve the mapped attribute values from the user profile first. If the attribute values are not present in the user's profile, then it attempts to retrieve them from the user's session.

Default: `com.sun.identity.saml2.plugins.DefaultIDPAttributeMapper`

### Attribute Map

Maps SAML attributes to user profile attributes.

The user profile attributes used here must both be allowed in user profiles, and also be specified for the identity repository. See the *Developer's Guide* chapter, *Customizing Profile Attributes* in the *Developer's Guide*, for instructions on allowing additional attributes in user profiles.

To specify the list of profile attributes for an LDAP identity repository, login to OpenAM Console as administrator and browse to Access Control > *Realm Name* > Data Stores, and click the data store name to open the configuration page. Scroll down to User Configuration, and edit the LDAP User Attributes list, and then click Save to keep your work.

The default IDP mapping implementation allows you to add static values in addition to values taken from the user profile. You add a static value by enclosing the profile attribute name in double quotes ("), as in the following examples.

To add a static SAML attribute called `nameID` with a value of `staticNameIDValue` with a name format of `urn:oasis:names:tc:SAML:2.0:attrname-format:uri`, add the following mapping.

```
urn:oasis:names:tc:SAML:2.0:attrname-format:uri|nameID="staticNameIDValue"
```

## Account Mapper

### Account Mapper

Specifies a class that implements `AccountMapper` to map remote users to local user profiles.

## Local Configuration

### Auth URL

URL where users are redirected to authenticate.

### Reverse Proxy URL

When a reverse proxy is used for SAML endpoints, it is specified here.

### External Application Logout URL

URL to which to send an HTTP POST including all cookies when receiving a logout request. To add a user session property as a POST parameter, include it in the URL query string as a `appsessionproperty` parameter.

## 12.3.3. Hints for Services

Use the following hints to adjust settings on the Services tab page.

### MetaAlias

#### MetaAlias

Used to locate the providers entity identifier, specified as `[/realm-name]*/provider-name`, where neither *realm-name* nor *provider-name* can contain slash characters (/). For example: `/realm/sub-realm/idp`.

### IDP Service Attributes

#### Artifact Resolution Service

Specifies the end point to handle artifact resolution. The Index is a unique number identifier for the end point.

#### Single Logout Service

Specifies the end points to handle single logout, depending on the SAML binding selected.

#### Manage NameID Service

Specifies the end points to handle name identifiers, depending on the SAML binding selected.

#### Single SignOn Service

Specifies the end points to handle single sign on.

## *NameID Mapping*

### **URL**

Specifies the end point to handle name identifier mapping.

## 12.3.4. Hints for Advanced Settings

Use the following hints to adjust settings on the Advanced tab page.

## *SAE Configuration*

### **IDP URL**

Specifies the end point to handle Secure Attribute Exchange requests.

### **Application Security Configuration**

Specifies how to handle encryption for Secure Attribute Exchange operations.

## *ECP Configuration*

### **IDP Session Mapper**

Specifies the class that finds a valid session from an HTTP servlet request to an identity provider with a SAML Enhanced Client or Proxy profile.

## *Session Synchronization*

### **Enabled**

When enabled, the identity provider notifies service providers to log the user out when a session expires.

## *IDP Finder Implementation*

### **IDP Finder Implementation Class**

Specifies a class that finds the preferred identity provider to handle a proxied authentication request.

### **IDP Finder JSP**

Specifies a JSP that presents the list of identity providers to the user.

## Enable Proxy IDP Finder For All SPs

When enabled, apply the finder for all remote service providers.

### *Relay State URL List*

#### **Relay State URL List**

List of URLs permitted for the `RelayState` parameter. OpenAM validates the redirection URL in the `RelayState` parameter against this list. If the `RelayState` parameter's value is in the list, OpenAM allows redirection to the `RelayState` URL. If it is not in the list, a browser error occurs.

If you do not specify any URLs in this property, OpenAM does not validate the `RelayState` parameter.

### *IDP Adapter*

#### **IDP Adapter Class**

Specifies a class to invoke immediately before sending a SAML 2.0 response.

## 12.4. Configuring Service Providers

Once you have set up a service provider, you can configure it through the OpenAM console under Federation > Entity Providers > *Provider Name*.

### 12.4.1. Hints for Assertion Content

Use the following hints to adjust settings on the Assertion Content tab page.

#### *Signing and Encryption*

##### **Request/Response Signing**

Specifies what parts of messages the service provider requires the identity provider to sign digitally.

##### **Encryption**

The identity provider must encrypt selected elements.

##### **Certificate Aliases**

Specifies aliases for certificates in the OpenAM key store that are used to handle digital signatures, and to handle encrypted messages.

## *NameID Format*

### **NameID Format List**

Specifies the supported name identifiers for users that are shared between providers for single sign on. If no name identifier is specified when initiating single sign on, then the service provider uses the first one in the list supported by the identity provider.

### **Disable Federation persistence if NameID Format is unspecified**

When enabled, the NameID Format in the authentication response is `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified`, and the Account Mapper has identified the local user, the service provider does not persist federation information in the user profile.

## *Authentication Context*

### **Mapper**

Specifies a class that implements the `SPAuthnContextMapper` interface and sets up the authentication context.

### **Default Authentication Context**

Specifies the authentication context used if no authentication context specified in the request.

### **Supported Contexts**

Specifies the supported authentication contexts. The Level corresponds to an authentication module authentication level.

### **Comparison Type**

How the authentication context in the assertion response must compare to the supported contexts.

## *Assertion Time*

### **Assertion Time Skew**

Grace period in seconds for the `NotBefore` time in assertions.

## *Basic Authentication*

### **Enabled, User Name, Password**

When enabled, authenticate with the specified user name and password at SOAP end points.

## 12.4.2. Hints for Assertion Processing

Use the following hints to adjust settings on the Assertion Processing tab page.

### *Attribute Mapper*

#### **Attribute Mapper**

Specifies a class that implements the attribute mapping.

#### **Attribute Map**

Maps SAML attributes to user profile attributes.

### *Auto Federation*

#### **Enabled**

When enabled, automatically federate user's accounts at different providers based on the specified profile attribute.

#### **Attribute**

Specifies the SAML profile attribute to match accounts at different providers.

### *Account Mapper*

#### **Account Mapper**

Specifies a class that implements `AccountMapper` to map remote users to local user profiles.

#### **Use Name ID as User ID**

When selected, fall back to using the name identifier from the assertion to find the user.

### *Artifact Message Encoding*

#### **Encoding**

Specifies the message encoding format for artifacts.

### *Transient User*

#### **Transient User**

Specifies the user profile to which to map all identity provider users when sending transient name identifiers.

## URL

### Local Authentication URL

Specifies the local login URL.

### Intermediate URL

Specifies a URL to which the user is redirected after authentication but before the original URL requested.

### External Application Logout URL

Specifies the URL to which to send an HTTP POST including all cookies when receiving a logout request. To add a user session property as a POST parameter, include it in the URL query string as a `appsessionproperty` parameter.

### Default Relay State URL

### Default Relay State URL

Specifies the URL to which to redirect users after the request has been handled. Used if not specified in the response.

## Adapter

### Adapter

Specifies a class that implements the `FederationSPAdapter` interface and performs application specific processing during the federation process.

### Adapter Environment

Specifies environment variables passed to the adapter class.

## 12.4.3. Hints for Services

Use the following hints to adjust settings on the Services tab page.

### MetaAlias

### MetaAlias

Used to locate the providers entity identifier, specified as `[/realm-name]*/provider-name`, where neither `realm-name` nor `provider-name` can contain slash characters (`/`). For example: `/realm/sub-realm/sp`.

## *SP Service Attributes*

### **Single Logout Service**

Specifies the end points to handle single logout, depending on the SAML binding selected.

### **Manage NameID Service**

Specifies the end points to handle name identifiers, depending on the SAML binding selected.

### **Assertion Consumer Service**

Specifies the end points to consume assertions, with Index corresponding to the index of the URL in the standard metadata.

## 12.4.4. Hints for Advanced Settings

Use the following hints to adjust settings on the Advanced tab page.

### *SAE Configuration*

#### **SP URL**

Specifies the end point to handle Secure Attribute Exchange requests.

#### **SP Logout URL**

Specifies the end point of the service provider that can handle global logout requests.

#### **Application Security Configuration**

Specifies how to handle encryption for Secure Attribute Exchange operations.

### *ECP Configuration*

#### **Request IDP List Finder Implementation**

Specifies a class that returns a list of preferred identity providers trusted by the SAML Enhanced Client or Proxy profile.

#### **Request IDP List Get Complete**

Specifies a URI reference used to retrieve the complete identity provider list if the `IDPList` element is not complete.

#### **Request IDP List**

Specifies a list of identity providers for the SAML Enhanced Client or Proxy to contact, used by the default implementation of the IDP Finder.

## *IDP Proxy*

### **IDP Proxy**

When enabled, allow proxied authentication for this service provider.

### **Introduction**

When enabled, use introductions to find the proxy identity provider.

### **Proxy Count**

Specifies the maximum number of proxy identity providers.

### **IDP Proxy List**

Specifies a list of URIs identifying preferred proxy identity providers.

## *Session Synchronization*

### **Enabled**

When enabled, the service provider notifies identity providers to log the user out when a session expires.

## *Relay State URL List*

### **Relay State URL List**

List of URLs permitted for the **RelayState** parameter. OpenAM validates the redirection URL in the **RelayState** parameter against this list. If the **RelayState** parameter's value is in the list, OpenAM allows redirection to the **RelayState** URL. If it is not in the list, a browser error occurs.

If you do not specify any URLs in this property, OpenAM does not validate the **RelayState** parameter.

## 12.5. Configuring Circles of Trust

Once you have set up a circle of trust, you can configure it through the OpenAM console under Federation > Circle of Trust > *Circle of Trust Name*.

### **Name**

String to refer to the circle of trust.

### **Description**

Short description of the circle of trust.

### IDFF Writer Service URL

Liberty Identity Federation Framework service that writes identity provider entity identifiers to Common Domain cookies after successful authentication, used in identity provider discovery.

Example: <http://www.disco.example:8080/openam/idffwriter>.

### IDFF Reader Service URL

Liberty Identity Federation Framework service that reads identity provider entity identifiers from Common Domain cookies, used in identity provider discovery. Example: [http://www.disco](http://www.disco.example:8080/openam/transfer)

[.example:8080/openam/transfer](http://www.disco.example:8080/openam/transfer).

### SAML2 Writer Service URL

SAML 2.0 service that writes identity provider entity identifiers to Common Domain cookies after successful authentication, used in identity provider discovery. Example: [http://www.disco](http://www.disco.example:8080/openam/saml2writer)

[.example:8080/openam/saml2writer](http://www.disco.example:8080/openam/saml2writer).

### SAML2 Reader Service URL

SAML 2.0 service that reads identity provider entity identifiers from Common Domain cookies, used in identity provider discovery. Example: <http://www.disco.example:8080/openam/saml2reader>.

### Status

Whether this circle of trust is operational.

### Realm

Name of the realm participating in this circle of trust.

### Entity Providers

Known hosted and remote identity and service providers participating in this circle of trust.

## 12.6. Configuring Google Apps as a Remote Service Provider

OpenAM can serve as the identity provider when you use [Google Apps](#) as a service provider, allowing users to have single sign-on with their Google Apps account.

In order to use this service, you must have a Google Apps account for at least one of your domains, such as [example.com](#).

### *Procedure 12.10. To Integrate With Google Apps*

1. If you have not yet done so, set up OpenAM as described in Procedure 12.1, "To Create a Hosted Identity Provider", using a signing certificate that is needed by Google Apps.

See the procedure *To Change the Signing Key for Federation* for details regarding the signing certificate.

2. On the OpenAM console Common Tasks page, click Configure Google Apps.
3. On the first Configure Google Apps for Single Sign-On page, add your domain name(s) such as `example.com` to the list, and then click Create.
4. On the second Configure Google Apps for Single Sign-On page, save the OpenAM verification certificate to a text file, such as `OpenAM.pem`.
5. Follow the instructions under To Enable Access to the Google Apps API before clicking Finish.
  - a. Access the Google Apps administration page for the first of your domains in a new browser tab or window.
  - b. Login as Google Apps administrator.
  - c. Select Enable Single Sign-On.
  - d. Copy the URLs from the OpenAM page into the Google Apps setup screen.
  - e. Upload the certificate file you saved such as `OpenAM.pem` as the Google Apps Verification Certificate.
  - f. Select Use a domain specific issuer.
  - g. Save changes in Google Apps setup.
  - h. Repeat the steps above for each domain you have configured.
  - i. Click Finish to complete the process.

## 12.7. Configuring Salesforce CRM as a Remote Service Provider

OpenAM can serve as the identity provider when you use Salesforce CRM as a service provider, allowing users to have single sign-on with their Salesforce CRM account.

In order to use this service, you must have Salesforce CRM accounts for your organization.

### *Procedure 12.11. To Integrate With Salesforce CRM*

1. If you have not yet done so, set up OpenAM as described in Procedure 12.1, "To Create a Hosted Identity Provider", using a signing certificate that is needed by Salesforce CRM.

See the procedure *To Change the Signing Key for Federation* for details regarding the signing certificate.

2. On the OpenAM console Common Tasks page, click Configure Salesforce CRM.

3. Enter the EntityID for your Salesforce service provider.

This ID is used as the persistent EntityDescriptor metadata element so that users can have multiple service provider instances. This field is used for the EntityDescriptor on the next page.

4. On the first Salesforce CRM Single Sign-On Configuration page, configure attribute mapping to associate the appropriate attribute from Salesforce CRM with the user profile attribute on your IDP.

For example, add a mapping for `IDPEmail` to `mail`, and then click Create. Make sure the attribute mapper is sending the correct attribute to be used for the federated identity.

5. On the second Salesforce CRM Single Sign-On Configuration page, follow the instructions below before clicking Finish.

- a. In a new browser tab or window, login to Salesforce CRM with your administrator credentials.

Create an administrator account if none exists, yet.

- b. If your users go directly to Salesforce to access services, then their single sign-on is SP-initiated from the Salesforce side. Salesforce provides a "My Domain" feature to facilitate SP-initiated single sign-on for desktop and device users.

When you have completed configuring Salesforce as a service provider, users can then browse to your domain at Salesforce, such as <https://openam.my.salesforce.com>, and be redirected to OpenAM to authenticate before being redirected to Salesforce.

- i. Select Administration Setup > Company Profile > My Domain.
- ii. Choose the domain name, and then register the domain.
- iii. Wait until the domain is ready for testing to proceed.
- c. In Salesforce CRM, browse to Setup > Administration Setup > Security Controls > Single Sign-On Settings, and then click Edit for Single Sign-On Settings.
- d. Select SAML Enabled.
- e. Set the SAML Version to 2.0.
- f. Copy the issuer name from the OpenAM page to the Issuer field on the Salesforce CRM page.
- g. Copy or download the OpenAM verification certificate to a text file, such as `OpenAMCert.pem` or `OpenAMCert.txt`.
- h. Upload the certificate file as Identity Provider Certificate on the Salesforce CRM page.
- i. For SAML Identity Type in Salesforce CRM, choose Assertion contains the Federation ID from the User object.

- j. For SAML Identity Location in Salesforce CRM, choose Identity is in an Attribute element.
  - k. If you require specific login or logout pages, enter them in the next two fields.
  - l. Enter the URL of your page specific error page if you have a page where you would like users redirected to when they encounter an error.
  - m. Copy the attribute name such as `IDPEmail` from the OpenAM page to the Attribute Name field on the Salesforce CRM page.
  - n. Leave the NameID Format field empty.
  - o. Select the Entity ID corresponding to the "My Domain" that you set up.
  - p. Save your work in Salesforce CRM.
  - q. Salesforce CRM displays a Salesforce Login URL.  
Copy the Salesforce Login URL to the field provided on the OpenAM page.
  - r. Salesforce CRM returns to the Single Sign-On Settings form.
  - s. Click Download Metadata to download the Salesforce CRM SP metadata.  
After you complete the configuration, you must import the SP metadata you download in this step.
  - t. In Salesforce CRM, browse to Administration Setup > Manage Users, and then click Users.
  - u. Add users as necessary, making sure the attribute chosen as the Federation ID matches the local attribute you mapped to the remote attribute in the previous page in OpenAM.
  - v. Click Finish to complete the process.
6. After you finish, import the metadata for Salesforce CRM as SP.
    - a. Browse in OpenAM console to the Federation tab.
    - b. If the remote SP entity for Salesforce CRM is already in the Entity Providers list, delete the existing configuration.
    - c. Click Import Entity..., and then use the Import Entity Provider page to import the Salesforce CRM metadata.
      - Update the `Realm Name` to the appropriate realm.
      - Select the location where the metadata file is.
      - Enter the path for the metadata file.

- If you have an extended data file, select the location where the file is.
- If you have an extended data file, enter the path for the metadata file.

At this point, when a user browses to the Salesforce domain you set up, they should be redirected to OpenAM for authentication. Upon successful authentication, they should be logged in to Salesforce.

## 12.8. Using SAML 2.0 Single Sign-On & Single Logout

OpenAM SAML 2.0 Federation provides JSPs where you can direct users to do single sign-on (SSO) and single logout (SLO) across providers in a circle of trust. OpenAM has two JSPs for SSO and two JSPs for SLO, allowing you to initiate both processes either from the identity provider side, or from the service provider side.

SSO lets users sign in once and remain authenticated as they access services in the circle of trust.

SLO attempts to log a user out of all providers in the circle of trust.

The JSP pages are found under the context root where you deployed OpenAM, in `saml2/jsp/`.

### `spSSOInit.jsp`

Used to initiate SSO from the service provider side, so call this on the service provider not the identity provider. This is also mapped to the endpoint `spsssoinit` under the context root.

Examples: `http://www.sp.example:8080/openam/saml2/jsp/spSSOInit.jsp`, `http://www.sp.example:8080/openam/spsssoinit`

### `idpSSOInit.jsp`

Used to initiate SSO from the identity provider side, so call this on the identity provider not the service provider. This is also mapped to the endpoint `idpsssoinit` under the context root.

Examples: `http://www.idp.example:8080/openam/saml2/jsp/idpSSOInit.jsp`, `http://www.idp.example:8080/openam/idpsssoinit`

### `spSingleLogoutInit.jsp`

Used to initiate SLO from the service provider side, so call this on the service provider not the identity provider.

Example: `http://www.sp.example:8080/openam/saml2/jsp/spSingleLogoutInit.jsp`, `http://www.sp.example:8080/openam/SPSloInit`

### `idpSingleLogoutInit.jsp`

Used to initiate SLO from the identity provider side, so call this on the identity provider not the service provider.

**Example:** `http://www.idp.example:8080/openam/saml2/jsp/idpSingleLogoutInit.jsp`, `http://www.idp.example:8080/openam/IDPSloInit`

When you invoke these JSPs, there are several parameters to specify. Which parameters you can use depends on the JSP.

### *idpSSOInit.jsp* Parameters

#### **metaAlias**

(Required) Use this parameter to specify the local alias for the provider, such as `metaAlias=/myRealm/idp`. This parameter takes the format `/realm-name/provider-name` as described in `MetaAlias`. You do not repeat the slash for the top level realm, for example `metaAlias=/idp`.

#### **spEntityID**

(Required) Use this parameter to indicate the remote service provider. Make sure you URL encode the value. For example, specify `spEntityID=http://www.sp.example:8080/openam` as `spEntityID=http%3A%2F%2Fwww.sp.example%3A8080%2Fopenam`.

#### **affiliationID**

(Optional) Use this parameter to specify a SAML affiliation identifier.

#### **binding**

(Optional) Use this parameter to indicate what binding to use for the operation. For example, specify `binding=HTTP-POST` to use HTTP POST binding with a self-submitting form. In addition to `binding=HTTP-POST`, you can also use `binding=HTTP-Artifact`.

#### **NameIDFormat**

(Optional) Use this parameter to specify a SAML Name Identifier format identifier such as `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent`, OR `urn:oasis:names:tc:SAML:2.0:nameid-format:transient`.

#### **RelayState**

(Optional) Use this parameter to specify where to redirect the user when the process is complete. Make sure you URL encode the value. For example, `RelayState=http%3A%2F%2Fforgerock.com` takes the user to `http://forgerock.com`.

#### **RelayStateAlias**

(Optional) Use this parameter to specify the parameter to use as the `RelayState`. For example, if your query string has `target=http%3A%2F%2Fforgerock.com&RelayStateAlias=target`, this is like setting `RelayState=http%3A%2F%2Fforgerock.com`.

## `spSSOInit.jsp` Parameters

### `idpEntityID`

(Required) Use this parameter to indicate the remote identity provider. Make sure you URL encode the value. For example, specify `idpEntityID=http://www.idp.example:8080/openam` as `idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam`.

### `metaAlias`

(Required) Use this parameter to specify the local alias for the provider, such as `metaAlias=/myRealm/sp`. This parameter takes the format `/realm-name/provider-name` as described in `MetaAlias`. You do not repeat the slash for the top level realm, `metaAlias=/sp`.

### `affiliationID`

(Optional) Use this parameter to specify a SAML affiliation identifier.

### `AllowCreate`

(Optional) Use this parameter to indicate whether the identity provider can create a new identifier for the principal if none exists (`true`) or not (`false`).

### `AssertionConsumerServiceIndex`

(Optional) Use this parameter to specify an integer that indicates the location to which the Response message should be returned to the requester.

### `AuthComparison`

(Optional) Use this parameter to specify a comparison method to evaluate the requested context classes or statements. OpenAM accepts the following values: `better`, `exact`, `maximum`, and `minimum`.

### `AuthnContextClassRef`

(Optional) Use this parameter to specify authentication context class references. Separate multiple values with pipe characters (`|`).

### `AuthnContextDeclRef`

(Optional) Use this parameter to specify authentication context declaration references. Separate multiple values with pipe characters (`|`).

### `AuthLevel`

(Optional) Use this parameter to specify the authentication level of the authentication context that OpenAM should use to authenticate the user.

### binding

(Optional) Use this parameter to indicate what binding to use for the operation. For example, specify `binding=HTTP-POST` to use HTTP POST binding with a self-submitting form. In addition to `binding=HTTP-POST`, you can also use `binding=HTTP-Artifact`.

### Destination

(Optional) Use this parameter to specify a URI Reference indicating the address to which the request is sent.

### ForceAuthn

(Optional) Use this parameter to indicate whether the identity provider should force authentication (`true`) or can reuse existing security contexts (`false`).

### isPassive

(Optional) Use this parameter to indicate whether the identity provider should authenticate passively (`true`) or not (`false`).

### NameIDFormat

(Optional) Use this parameter to specify a SAML Name Identifier format identifier such as `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent`, or `urn:oasis:names:tc:SAML:2.0:nameid-format:transient`.

### RelayState

(Optional) Use this parameter to specify where to redirect the user when the process is complete. Make sure you URL encode the value. For example, `RelayState=http%3A%2F%2Fforgerock.com` takes the user to `http://forgerock.com`.

### RelayStateAlias

(Optional) Use this parameter to specify the parameter to use as the `RelayState`. For example, if your query string has `target=http%3A%2F%2Fforgerock.com&RelayStateAlias=target`, this is like setting `RelayState=http%3A%2F%2Fforgerock.com`.

### reqBinding

(Optional) Use this parameter to indicate what binding to use for the authentication request. Valid values include `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect` (default) and `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST`.

### sunamcompositeadvice

(Optional) Use this parameter to specify a URL encoded XML blob that specifies the authentication level advice. For example, the following XML indicates a requested authentication level of 1. Notice the required `&` before the 1.

```
<Advice>
  <AttributeValuePair>
    <Attribute name="AuthLevelConditionAdvice"/>
    <Value>/:1</Value>
  </AttributeValuePair>
</Advice>
```

## *idpSingleLogoutInit.jsp* Parameters

### **binding**

(Required) Use this parameter to indicate what binding to use for the operation. The full, long name format is required for this parameter to work.

The value must be one of the following.

- `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect` (default)
- `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST`
- `urn:oasis:names:tc:SAML:2.0:bindings:SOAP`

### **Consent**

(Optional) Use this parameter to specify a URI that is a SAML Consent Identifier.

### **Destination**

(Optional) Use this parameter to specify a URI Reference indicating the address to which the request is sent.

### **Extension**

(Optional) Use this parameter to specify a list of Extensions as string objects.

### **goto**

(Optional) Use this parameter to specify where to redirect the user when the process is complete. `RelayState` takes precedence over this parameter.

### **LogoutAll**

(Optional) Use this parameter to specify that the identity provider should send single logout requests to service providers without indicating a session index.

### **RelayState**

(Optional) Use this parameter to specify where to redirect the user when the process is complete. Make sure you URL encode the value. For example, `RelayState=http%3A%2F%2Fforgerock.com` takes the user to `http://forgerock.com`.

## `spSingleLogoutInit.jsp` Parameters

### **binding**

(Required) Use this parameter to indicate what binding to use for the operation. The full, long name format is required for this parameter to work. For example, specify `binding=urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST` to use HTTP POST binding with a self-submitting form rather than the default HTTP redirect binding. In addition, you can use `binding=urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact`.

### **idpEntityID**

(Required for Fedlets) Use this parameter to indicate the remote identity provider. If the `binding` is not set, then OpenAM uses this parameter to find the default binding. Make sure you URL encode the value. For example, specify `idpEntityID=http://www.sp.example:8080/openam` as `idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam`.

### **NameIDValue**

(Required for Fedlets) Use this parameter to indicate the SAML Name Identifier for the user.

### **SessionIndex**

(Required for Fedlets) Use this parameter to indicate the `sessionIndex` of the user session to terminate.

### **Consent**

(Optional) Use this parameter to specify a URI that is a SAML Consent Identifier.

### **Destination**

(Optional) Use this parameter to specify a URI Reference indicating the address to which the request is sent.

### **Extension**

(Optional) Use this parameter to specify a list of Extensions as string objects.

### **goto**

(Optional) Use this parameter to specify where to redirect the user when the process is complete. `RelayState` takes precedence over this parameter.

### **RelayState**

(Optional) Use this parameter to specify where to redirect the user when the process is complete. Make sure you URL encode the value. For example, `RelayState=http%3A%2F%2Fforgerock.com` takes the user to `http://forgerock.com`.

### spEntityID

(Optional, for Fedlets) Use this parameter to indicate the Fedlet entity ID. When missing, OpenAM uses the first entity ID in the metadata.

#### Example 12.1. SSO & SLO From the Service Provider

The following URL takes the user from the service provider side to authenticate at the identity provider and then come back to the end user profile page at the service provider after successful SSO. Lines are folded to show you the query string parameters.

```
http://www.sp.example:8080/openam/saml2/jsp/spSSOInit.jsp?metaAlias=/sp
&idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam
&RelayState=http%3A%2F%2Fwww.sp.example%3A8080%2Fopenam%2Fidm%2FEndUser
```

The following URL initiates SLO from the service provider side, leaving the user at <http://forgerock.com>.

```
http://www.sp.example:8080/openam/saml2/jsp/spSingleLogoutInit.jsp?
&idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam
&RelayState=http%3A%2F%2Fforgerock.com
```

#### Procedure 12.12. To Indicate Progress During SSO

During SSO log in, OpenAM presents users with a self-submitting form when access has been validated. This page is otherwise blank. If you want to present users with something to indicate that the operation is in progress, then customize the necessary templates.

1. Modify the templates to add a clue that SSO is in progress, such as an image.

Edit the templates found in OpenAM sources, [saml2login.template](#) and [saml2loginwithrelay.template](#).

When you add an image or other presentation element, make sure that you retain the form and JavaScript as is. Also, as the files are templates, what you add must be static HTML.

2. Unpack OpenAM-11.0.0.war, and add your modified template files under `WEB-INF/classes/` where you unpacked the .war.

Also include any images you reference in the page.

3. Pack up your custom version of OpenAM, and then deploy it in your web container.

## 12.9. Configuring OpenAM For the ECP Profile

The SAML 2.0 Enhanced Client or Proxy (ECP) profile is intended for use when accessing services over devices like simple phones, medical devices, and set-top boxes that lack the capabilities needed to use the more widely used SAML 2.0 Web Browser SSO profile.

The ECP knows which identity provider to contact for the user, and is able to use the reverse SOAP (PAOS) SAML 2.0 binding for the authentication request and response. The PAOS binding uses HTTP and SOAP headers to pass information about processing SOAP requests and responses, starting with a PAOS HTTP header that the ECP sends in its initial request to the server. The PAOS messages continue with a SOAP authentication request in the server's HTTP response to the ECP's request for a resource, followed by a SOAP response in an HTTP request from the ECP.

An enhanced client, such as a browser with a plugin or an extension, can handle these communications on its own. An enhanced proxy is an HTTP server such as a WAP gateway that can support the ECP profile on behalf of client applications.

OpenAM supports the SAML 2.0 ECP profile on the server side for identity providers and service providers. You must build the ECP.

By default an OpenAM identity provider uses the `com.sun.identity.saml2.plugins.DefaultIDPECPSessionMapper` class to find a user session for requests to the IDP from the ECP. The default session mapper uses OpenAM cookies as it would for any other client application. If for some reason you must change the mapping after writing and installing your own session mapper, you can change the class under Federation > Entity Providers > *idp-name* > IDP > Advanced > ECP Configuration.

By default an OpenAM service provider uses the `com.sun.identity.saml2.plugins.ECPIDPFinder` class to return identity providers from the list under Federation > Entity Providers > *sp-name* > SP > Advanced > ECP Configuration > Request IDP List. You must populate the list with identity provider entity IDs.

The endpoint for the ECP to contact on the OpenAM service provider is `/SPEC` as in `http://www.sp.example:8080/openam/SPEC`. The ECP provides two query string parameters to identify the service provider and to specify the URL of the resource to access.

#### `metaAlias`

This specifies the service provider, by default `metaAlias=/realm-name/sp`, as described in MetaAlias.

#### `RelayState`

This specifies the resource the client aims to access such as `RelayState=http%3A%2F%2Fforgerock.org%2Findex.html`.

For example, the URL to access the service provider and finally the resource at `http://forgerock.org/index.html` could be `http://www.sp.example:8080/openam/SPEC?metaAlias=/sp&RelayState=http%3A%2F%2Fforgerock.org%2Findex.html`.

## 12.10. Managing Federated Accounts

Identity providers and service providers must be able to communicate about users. Yet in some cases the identity provider can choose to communicate a minimum of information about an authenticated

user, with no user account maintained on the service provider side. In other cases the identity provider and service provider can choose to link user accounts in a persistent way, in a more permanent way, or even in automatic fashion by using some shared value in the user's profiles such as an email address or by dynamically creating accounts on the service provider when necessary. OpenAM supports all these alternatives.

### 12.10.1. Using Transient Federation Identifiers

OpenAM allows you to link accounts using transient name identifiers, where the identity provider shares a temporary identifier with the service provider for the duration of the user session. Nothing is written to the user profile.

Transient identifiers are useful where the service is anonymous, and all users have similar access on the service provider side.

To use transient name identifiers, specify the name ID format `urn:oasis:names:tc:SAML:2.0:nameid-format:transient` when initiating single sign on.

The examples below work in an environment where the identity provider is `www.idp.example` and the service provider is `www.sp.example`. Both providers have deployed OpenAM on port 8080 under deployment URI `/openam`.

To initiate single sign on from the service provider, access the following URL with at least the query parameters shown.

```
http://www.sp.example:8080/openam/saml2/jsp/spSSOInit.jsp?
  idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam
  &metaAlias=/sp
  &NameIDFormat=urn:oasis:names:tc:SAML:2.0:nameid-format:transient
```

For a complete list of query parameters, see `spSSOInit.jsp` Parameters.

To initiate single sign on from the identity provider, access the following URL with at least the query parameters shown.

```
http://www.idp.example:8080/openam/saml2/jsp/idpSSOInit.jsp?
  spEntityID=http%3A%2F%2Fwww.sp.example%3A8080%2Fopenam
  &metaAlias=/idp
  &NameIDFormat=urn:oasis:names:tc:SAML:2.0:nameid-format:transient
```

For a complete list of query parameters, see `idpSSOInit.jsp` Parameters.

The accounts are only linked for the duration of the session. Once the user logs out for example the accounts are no longer linked.

### 12.10.2. Using Persistent Federation Identifiers

OpenAM lets you use persistent pseudonym identifiers to federate user identities, linking accounts on the identity provider and service provider with a SAML persistent identifier.

Persistent identifiers are useful for establishing links between otherwise unrelated accounts.

The examples below work in an environment where the identity provider is [www.idp.example](http://www.idp.example) and the service provider is [www.sp.example](http://www.sp.example). Both providers have deployed OpenAM on port 8080 under deployment URI `/openam`.

To initiate single sign on from the service provider, access the following URL with at least the query parameters shown.

```
http://www.sp.example:8080/openam/saml2/jsp/spSSOInit.jsp?
idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam
&metaAlias=/sp
&NameIDFormat=urn:oasis:names:tc:SAML:2.0:nameid-format:persistent
```

For a complete list of query parameters, see [spSSOInit.jsp Parameters](#).

To initiate single sign on from the identity provider, access the following URL with at least the query parameters shown.

```
http://www.idp.example:8080/openam/saml2/jsp/idpSSOInit.jsp?
spEntityID=http%3A%2F%2Fwww.sp.example%3A8080%2Fopenam
&metaAlias=/idp
&NameIDFormat=urn:oasis:names:tc:SAML:2.0:nameid-format:persistent
```

For a complete list of query parameters, see [idpSSOInit.jsp Parameters](#).

On successful login, the accounts are persistently linked, with persistent identifiers stored in the user's accounts on the identity provider and the service provider.

### 12.10.3. Changing Federation of Persistently Linked Accounts

OpenAM implements the SAML 2.0 Name Identifier Management profile, allowing you to change a persistent identifier that has been set to federate accounts, and also to terminate federation for an account.

When user accounts are stored in an LDAP directory server, name identifier information is stored on the `sun-fm-saml2-nameid-info` and `sun-fm-saml2-nameid-infokey` attributes of a user's entry.<sup>1</sup> You can retrieve the name identifier value by checking the value of `sun-fm-saml2-nameid-infokey`.

For example, if the user's entry in the directory shows `sun-fm-saml2-nameid-infokey: http://www.idp.example:8080/openam|http://www.sp.example:8080/openam|XyffEsr6Vixbnt0BSqIgLlFMGjR2`, then the name identifier is `XyffEsr6Vixbnt0BSqIgLlFMGjR2`.

You can use this identifier to initiate a change request from the service provider as in the following example.

```
http://www.sp.example:8080/openam/saml2/jsp/spMNIRRequestInit.jsp?
idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam
```

<sup>1</sup>These attribute types are configurable in the OpenAM console under Configuration > Global > SAMLv2 Service Configuration.

```
&metaAlias=/sp
&requestType=NewID
&IDPProvidedID=XyffEsR6Vixbnt0BSqIgLlFMGjR2
```

If desired, you can substitute `openam/SPMniInit` for `openam/saml2/jsp/spMNIRequestInit.jsp`

You can also initiate the change request from the identity provider as in the following example.

```
http://www.idp.example:8080/openam/saml2/jsp/idpMNIRequestInit.jsp?
spEntityID=http%3A%2F%2Fwww.sp.example%3A8080%2Fopenam
&metaAlias=/idp
&requestType=NewID
&SPProvidedID=XyffEsR6Vixbnt0BSqIgLlFMGjR2
```

If desired, you can substitute `openam/IDPMniInit` for `openam/saml2/jsp/idpMNIRequestInit.jsp`

### *idpMNIRequestInit.jsp* Parameters

#### **spEntityID**

(Required) Use this parameter to indicate the remote service provider. Make sure you URL encode the value. For example, specify `spEntityID=http://www.sp.example:8080/openam` as `spEntityID=http%3A%2F%2Fwww.sp.example%3A8080%2Fopenam`.

#### **metaAlias**

(Required) Use this parameter to specify the local alias for the provider, such as `metaAlias=/myRealm/idp`. This parameter takes the format `/realm-name/provider-name` as described in `MetaAlias`. You do not repeat the slash for the top level realm, for example `metaAlias=/idp`.

#### **requestType**

(Required) Type of manage name ID request, either `NewID` to change the ID, or `Terminate` to remove the information that links the accounts on the identity provider and service provider.

#### **SPProvidedID**

(Required if `requestType=NewID`) Name identifier in use as described above.

#### **affiliationID**

(Optional) Use this parameter to specify a SAML affiliation identifier.

#### **binding**

(Optional) Use this parameter to indicate what binding to use for the operation. The full, long name format is required for this parameter to work.

The value must be one of the following.

- `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST`

- `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect`
- `urn:oasis:names:tc:SAML:2.0:bindings:SOAP`

### **relayState**

(Optional) Use this parameter to specify where to redirect the user when the process is complete. Make sure you URL encode the value. For example, `relayState=http%3A%2F%2Fforgerock.com` takes the user to `http://forgerock.com`.

### *spMNIRquestInit.jsp* Parameters

#### **idpEntityID**

(Required) Use this parameter to indicate the remote identity provider. Make sure you URL encode the value. For example, specify `idpEntityID=http://www.idp.example:8080/openam` as `idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam`.

#### **metaAlias**

(Required) Use this parameter to specify the local alias for the provider, such as `metaAlias=/myRealm/sp`. This parameter takes the format `/realm-name/provider-name` as described in `MetaAlias`. You do not repeat the slash for the top level realm, `metaAlias=/sp`.

#### **requestType**

(Required) Type of manage name ID request, either `NewID` to change the ID, or `Terminate` to remove the information that links the accounts on the identity provider and service provider.

#### **IDPProvidedID**

(Required if `requestType=NewID`) Name identifier in use as described above.

#### **affiliationID**

(Optional) Use this parameter to specify a SAML affiliation identifier.

#### **binding**

(Optional) Use this parameter to indicate what binding to use for the operation. The full, long name format is required for this parameter to work.

The value must be one of the following.

- `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST`
- `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect`
- `urn:oasis:names:tc:SAML:2.0:bindings:SOAP`

## relayState

(Optional) Use this parameter to specify where to redirect the user when the process is complete. Make sure you URL encode the value. For example, `relayState=http%3A%2F%2Fforgerock.com` takes the user to <http://forgerock.com>.

You can terminate federation as described in Section 12.10.4, "Terminating Federation of Persistently Linked Accounts".

### 12.10.4. Terminating Federation of Persistently Linked Accounts

OpenAM lets you terminate account federation, where the accounts have been linked with a persistent identifier as described in Section 12.10.2, "Using Persistent Federation Identifiers".

The examples below work in an environment where the identity provider is `www.idp.example` and the service provider is `www.sp.example`. Both providers have deployed OpenAM on port 8080 under deployment URI `/openam`.

To initiate the process of terminating account federation from the service provider, access the following URL with at least the query parameters shown.

```
http://www.sp.example:8080/openam/saml2/jsp/spMNIRequestInit.jsp?  
idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam  
&metaAlias=/sp  
&requestType=Terminate
```

To initiate the process of terminating account federation from the identity provider, access the following URL with at least the query parameters shown.

```
http://www.idp.example:8080/openam/saml2/jsp/idpMNIRequestInit.jsp?  
spEntityID=http%3A%2F%2Fwww.sp.example%3A8080%2Fopenam  
&metaAlias=/idp  
&requestType=Terminate
```

### 12.10.5. Configuring Auto-Federation

OpenAM lets you configure the service provider to link an account based on an attribute value from the identity provider. When you know the user accounts on both the identity provider and the service provider share a common attribute value, such as an email address or other unique user identifier, you can use this method to link accounts without user interaction. See Procedure 12.13, "To Auto-Federate Accounts Based on an Attribute Value".

OpenAM also lets you map users on the identity provider temporarily to a single anonymous user account on the service provider, in order to exchange attributes about the user without a user-specific account on the service provider. This approach can be useful when the service provider either needs no user-specific account to provide a service, or when you do not want to retain a user profile on the service provider but instead you make authorization decisions based on attribute values from the identity provider. See Procedure 12.14, "To Auto-Federate Using a Single Service Provider Account".

OpenAM further allows you to use attributes from the identity provider to create accounts dynamically on the service provider. When using this method, you should inform the user and obtain consent to create the account if necessary. See Procedure 12.15, "To Auto-Federate With Dynamic Service Provider Account Creation".

### *Procedure 12.13. To Auto-Federate Accounts Based on an Attribute Value*

The following steps demonstrate how to auto-federate accounts based on an attribute value that is the same in both accounts.

Perform the following steps on the hosted identity provider(s), and again on the hosted service provider(s).

1. Login to the OpenAM console as administrator.
2. Browse to Federation > *hosted-provider-name* > Assertion Processing.
3. (Optional) If the attribute to use for auto-federation is not yet in the attribute map, add the attribute mapping, and then Save your work.
4. On the hosted service provider, under Auto Federation, select Enabled and enter the local attribute name in the Attribute field, and then Save your work.

### *Procedure 12.14. To Auto-Federate Using a Single Service Provider Account*

The following steps demonstrate how to auto-federate using a single anonymous user account on the service provider.

Perform the following steps on the hosted identity provider(s), and again on the hosted service provider(s).

1. Login to the OpenAM console as administrator.
2. Browse to Federation > *hosted-provider-name* > Assertion Processing.
3. (Optional) If you want to get attributes from the identity provider and the attributes are is not yet in the attribute map, add the attribute mapping, and then Save your work.
4. On the hosted service provider, under Transient User, set the single account to which to map all users, such as `anonymous`, and then Save your work.
5. After completing configuration on the providers, use transient identifiers to federate as described in Section 12.10.1, "Using Transient Federation Identifiers".

### *Procedure 12.15. To Auto-Federate With Dynamic Service Provider Account Creation*

The following steps demonstrate how to auto-federate, dynamically creating an account on the service provider if necessary.

1. Set up auto-federation as described in Procedure 12.13, "To Auto-Federate Accounts Based on an Attribute Value". The attributes you map from the identity provider are those that the service provider sets on the dynamically created accounts.
2. On the service provider console, browse to Access Control > *realm-name* > Authentication > All Core Settings..., and Dynamic or Dynamic with User Alias, which are described in *Hints For the Core Authentication Module*, and then Save your work.
3. To test your work, create a user on the identity provider, log out of the console, and initiate SSO logging in as the user you created.

To initiate SSO, browse to one of the OpenAM SAML 2.0 JSPs with the appropriate query parameters. The following is an example URL for service provider initiated SSO.

```
http://www.sp.example:8080/openam/saml2/jsp/spSSOInit.jsp?  
idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam  
&metaAlias=/sp
```

On success, check <http://www.sp.example:8080/openam/idm/EndUser> to see the new user account.

## 12.10.6. Linking Federated Accounts in Bulk

If you manage both the identity provider and service provider, you can link accounts out-of-band, in bulk. You make permanent connections for a list of identity provider and service provider by using the **ssoadm** bulk federation commands.

Before you can run the bulk federation commands, first establish the relationship between accounts, set up the providers as described in Section 12.2, "Setting Up SAML 2.0 SSO", and install the **ssoadm** command as described in *To Set Up Administration Tools* in the *Installation Guide*.

To understand the relationships between accounts, consider an example where the identity provider is at [idp.example.org](http://idp.example.org) and the service provider is at [sp.example.com](http://sp.example.com). A demo user account has the Universal ID, `id=demo,ou=user,dc=example,dc=org`, on the identity provider. That maps to the Universal ID, `id=demo,ou=user,dc=example,dc=com`, on the service provider.

The **ssoadm** command then needs a file that maps local user IDs to remote user IDs, one per line, separated by the vertical bar character `|`. Each line of the file appears as follows.

```
local-user-ID|remote-user-ID
```

In the example, starting on the service provider side, the line for the demo user reads as follows.

```
id=demo,ou=user,dc=example,dc=com|id=demo,ou=user,dc=example,dc=org
```

All the users' accounts mapped in your file must exist at the identity provider and the service provider when you run the commands to link them.

Link the accounts using the **ssoadm** bulk federation commands.

1. Prepare the data with the **ssoadm do-bulk-federation** command.

The following example starts on the service provider side.

```
$ cat /tmp/user-map.txt
id=demo,ou=user,dc=example,dc=com|id=demo,ou=user,dc=example,dc=org
$ ssoadm do-bulk-federation --metaalias /sp
--remoteentityid http://idp.example.org:8080/openam
--useridmapping /tmp/user-map.txt
--nameidmapping /tmp/name-map.txt
--adminid amadmin --password-file /tmp/pwd.txt
--spec saml2

Bulk Federation for this host was completed. To complete the
federation, name Id mapping file should be loaded to remote
provider.
```

2. Copy the name ID mapping output file to the other provider.

```
$ scp /tmp/name-map.txt openam@idp.example.org:/tmp/name-map.txt
openam@idp.example.org's password:
name-map.txt          100% 177    0.2KB/s   00:00
```

3. Import the name ID mapping file with the **ssoadm import-bulk-fed-data** command.

The following example is performed on the identity provider side.

```
$ ssoadm import-bulk-fed-data
--adminid amadmin --password-file /tmp/pwd.txt
--metaalias /idp --bulk-data-file /tmp/name-map.txt

Bulk Federation for this host was completed.
```

At this point the accounts are linked.

## 12.11. Using SAML 2.0 Artifacts

By default OpenAM transmits SAML messages by value. This makes it possible to access the SAML messages in the user agent. You can instead request that OpenAM transmit SAML messages by reference using SAML artifacts, which are small values that reference a SAML message. Providers then communicate directly to resolve artifacts, rather than sending the messages through the user agent.

When initiating single sign-on using `idpSS0Init.jsp` or `spSS0Init.jsp` for example, add `binding=HTTP-Artifact` to the list of query parameters. The following example works in an environment where the identity provider is `www.idp.example` and the service provider is `www.sp.example`. Both providers have deployed OpenAM on port 8080 under deployment URI `/openam`.

```
http://www.sp.example:8080/openam/saml2/jsp/spSS0Init.jsp?
idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam
&metaAlias=/sp
&binding=HTTP-Artifact
```

## Chapter 13

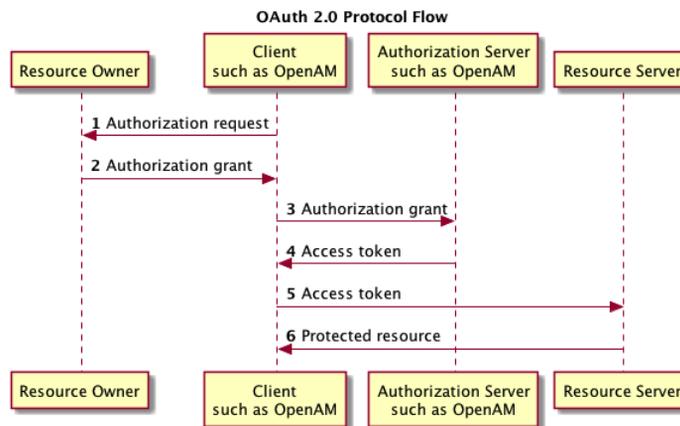
## Managing OAuth 2.0 Authorization

This chapter covers OpenAM support for the OAuth 2.0 authorization framework. The chapter begins by showing where OpenAM fits into the OAuth 2.0 authorization framework, and then shows how to configure the functionality.

## 13.1. About OAuth 2.0 Support in OpenAM

RFC 6749, *The OAuth 2.0 Authorization Framework*, provides a standard way for *resource owners* to grant *client* applications access to the owners' web-based resources. The canonical example involves a user (resource owner) granting access to a printing service (client) to print photos that the user has stored on a photo-sharing server.

The section describes how OpenAM supports the OAuth 2.0 authorization framework in terms of the roles that OpenAM plays.<sup>1</sup> The following sequence diagram indicates the primary roles OpenAM can play in the OAuth 2.0 protocol flow.



## 13.1.1. OpenAM as OAuth 2.0 Authorization Server

OpenAM can function as an OAuth 2.0 *authorization server*. In this role, OpenAM authenticates resource owners and obtains their authorization in order to return access tokens to clients.

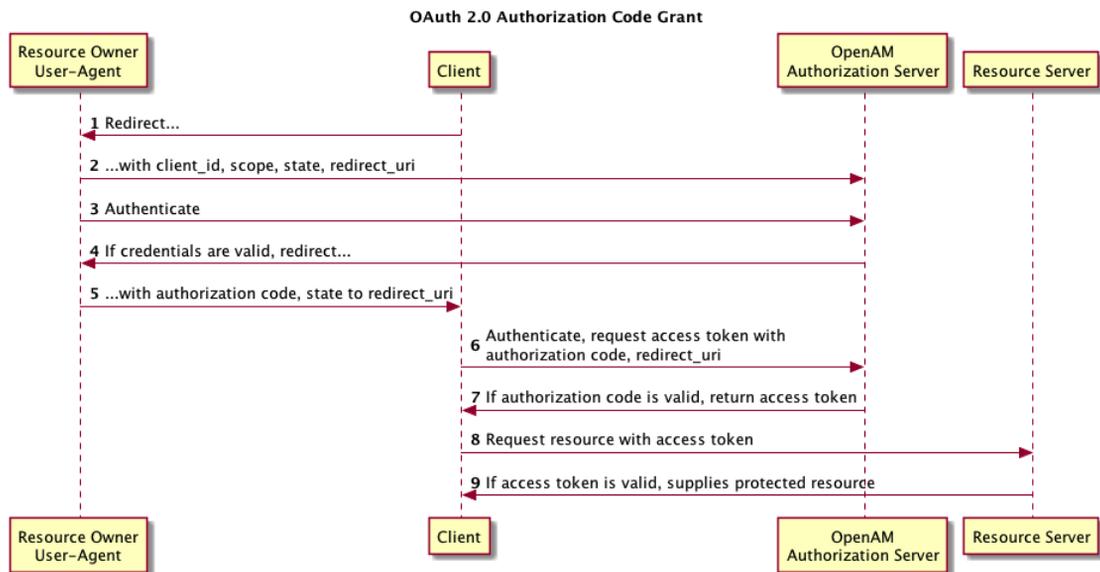
<sup>1</sup>Read RFC 6749 to understand the authorization framework itself.

When using OpenAM as authorization server, you register clients with OpenAM by using OAuth 2.0 policy agent profiles. OpenAM supports both confidential and public clients.

OpenAM supports the four main grants for obtaining authorization described in RFC 6749: the authorization code grant, the implicit grant, the resource owner password credentials grant, and the client credentials grant. See RFC 6749 for details on the authorization grant process, and for details on how clients should make authorization requests and handle authorization responses. OpenAM also supports the *SAML 2.0 Bearer Assertion Profiles for OAuth 2.0*, described in the Internet-Draft.

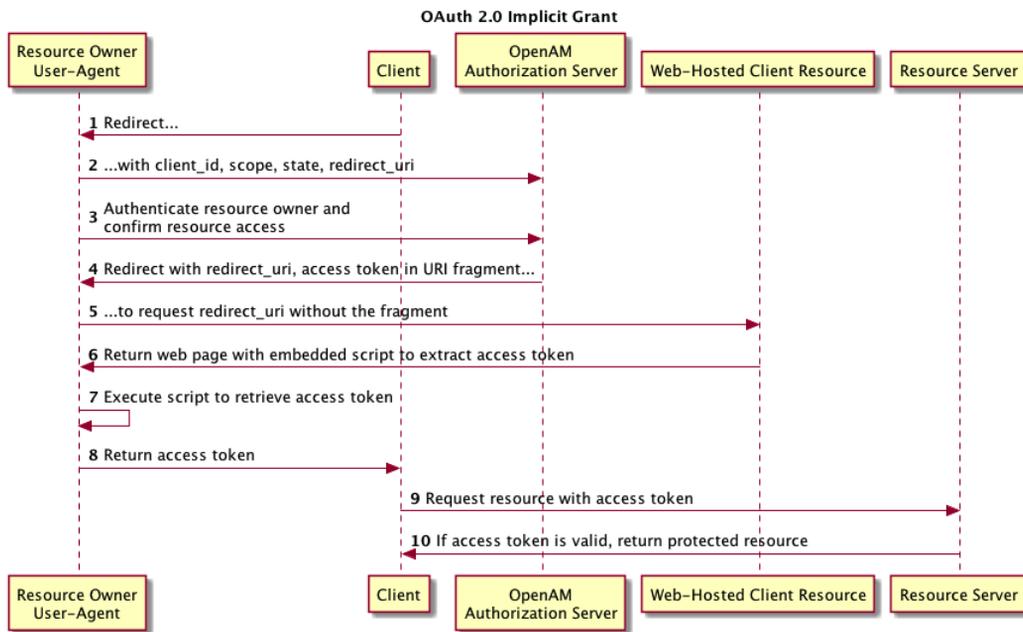
### 13.1.1.1. OAuth 2.0 Authorization Grant

The authorization code grant starts with the client, such as a web-based service, redirecting the resource owner's user-agent to the OpenAM authorization service. After authenticating the resource owner and obtaining the resource owner's authorization, OpenAM redirects the resource owner's user-agent back to the client with an authorization code that the client uses to request the access token. The following sequence diagram outlines a successful process from initial client redirection through to the client accessing the protected resource.



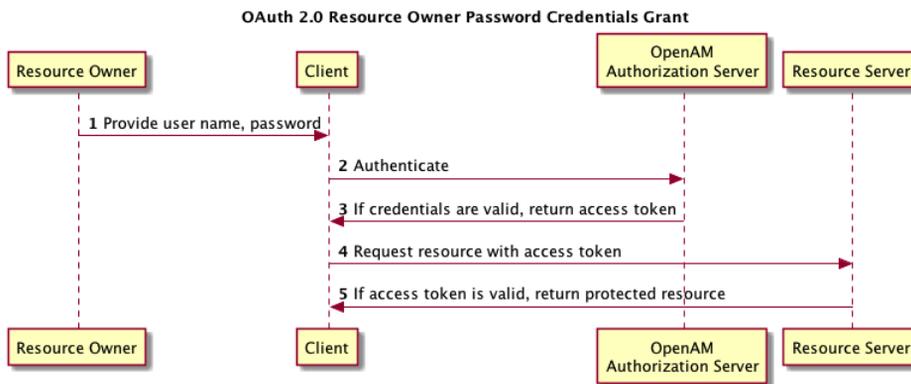
### 13.1.1.2. OAuth 2.0 Implicit Grant

The implicit grant is designed for clients implemented to run inside the resource-owner user agent. Instead of providing an authorization code that the client must use to retrieve an access token, OpenAM returns the access token directly in the fragment portion of the redirect URI. The following sequence diagram outlines the successful process.



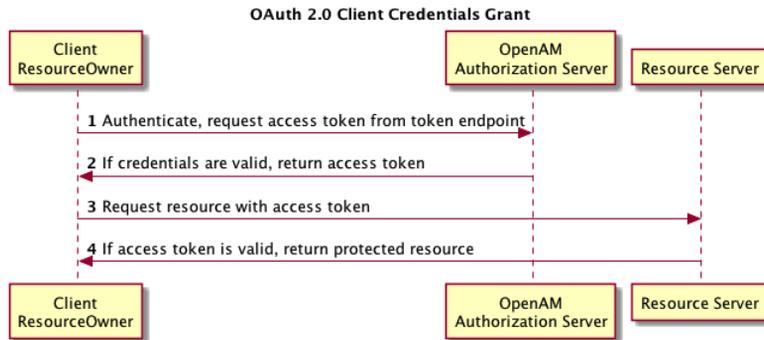
### 13.1.1.3. OAuth 2.0 Resource Owner Password Credentials Grant

The resource owner password credentials grant lets the client use the resource owner's user name and password to get an access token directly. Although this grant might seem to conflict with an original OAuth goal of not having to share resource owner credentials with the client, it can make sense in a secure context where other authorization grant types are not available, such as a client that is part of a device operating system using the resource owner credentials once and thereafter using refresh tokens to continue accessing resources. The following sequence diagram shows the successful process.



### 13.1.1.4. OAuth 2.0 Client Credentials Grant

The client credentials grant uses client credentials as an authorization grant. This grant makes sense when the client is also the resource owner, for example. The following sequence diagram shows the successful process.



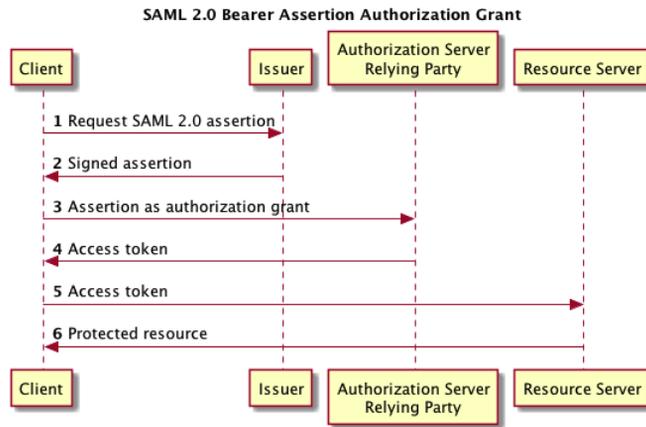
### 13.1.1.5. SAML 2.0 Bearer Assertion Profiles

The Internet-Draft, *SAML 2.0 Bearer Assertion Profiles for OAuth 2.0*, describes a means to use SAML 2.0 assertions to request access tokens and to authenticate OAuth 2.0 clients.

At present OpenAM implements the profile to request access tokens.

In both profiles, the issuer must sign the assertion. The client communicates the assertion over a channel protected with transport layer security, by performing an HTTP POST to the OpenAM's access token endpoint. OpenAM as OAuth 2.0 authorization server uses the issuer ID to validate the signature on the assertion.

In the profile to request an access token, the OAuth 2.0 client bears a SAML 2.0 assertion that was issued to the resource owner on successful authentication. A valid assertion in this case is equivalent to an authorization grant by the resource owner to the client. OAuth 2.0 clients must make it clear to the resource owner that by authenticating to the identity provider who issues the assertion, they are granting the client permission to access the protected resources.



The HTTP POST to OpenAM to request an access token looks something like this:

```

POST /openam/oauth2/access_token HTTP/1.1
Host: openam.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Asaml2-bearer&
assertion=PHNhbWxw0l...[base64url encoded assertion]...ZT4&
client_id=[ID registered with OpenAM]
    
```

If OpenAM is already a SAML 2.0 service provider, you can configure OpenAM as OAuth 2.0 authorization server as well, and set an adapter class name in the service provider configuration that lets OpenAM POST the assertion from the service provider to the authorization server. See Section 13.6, "Configuring OpenAM as Both SAML 2.0 Service Provider & OAuth 2.0 Authorization Server" for details.

### 13.1.1.6. OpenAM OAuth 2.0 Endpoints

In addition to the standard authorization and token endpoints described in RFC 6749, OpenAM also exposes a token information endpoint for resource servers to get information about access tokens so they can determine how to respond to requests for protected resources. OpenAM as authorization server exposes the following endpoints for clients and resource servers.

#### **/oauth2/authorize**

Authorization endpoint defined in RFC 6749, used to obtain an authorization grant from the resource owner

Example: <https://openam.example.com:8443/openam/oauth2/authorize>

#### **/oauth2/access\_token**

Token endpoint defined in RFC 6749, used to obtain an access token from the authorization server

Example: [https://openam.example.com:8443/openam/oauth2/access\\_token](https://openam.example.com:8443/openam/oauth2/access_token)

### [/oauth2/tokeninfo](#)

Endpoint not defined in RFC 6749, used to validate tokens, and to retrieve information such as scopes

Given an access token, a resource server can perform an HTTP GET on [/oauth2/tokeninfo?access\\_token=token-id](#) to retrieve a JSON object indicating `token_type`, `expires_in`, `scope`, and the `access_token` ID.

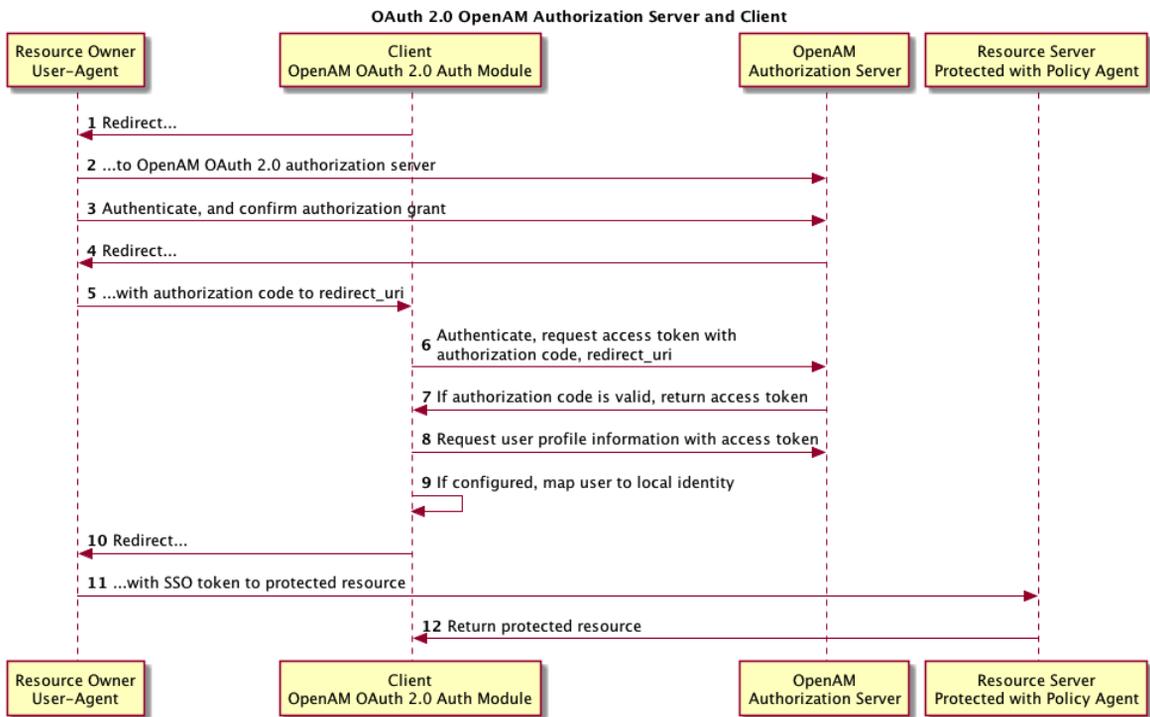
Example: <https://openam.example.com:8443/openam/oauth2/tokeninfo>

For examples, see the *Developer's Guide* section, *OAuth 2.0 Authorization* in the *Developer's Guide*.

## 13.1.2. OpenAM as OAuth 2.0 Client & Resource Server Solution

OpenAM can function as an OAuth 2.0 client for installations where the web resources are protected by OpenAM. To configure OpenAM as an OAuth 2.0 client, you set up an OpenAM OAuth 2.0 authentication module instance, and then integrate the authentication module into your authentication chains as necessary.

When OpenAM functions as an OAuth 2.0 client, OpenAM provides an OpenAM SSO session after successfully authenticating the resource owner and obtaining authorization. This means the client can then access resources protected by policy agents. In this respect the OpenAM OAuth 2.0 client is just like any other authentication module, one that relies on an OAuth 2.0 authorization server to authenticate the resource owner and obtain authorization. The following sequence diagram shows how the client gains access to protected resources in the scenario where OpenAM functions as both authorization server and client for example.



As the OAuth 2.0 client functionality is implemented as an OpenAM authentication module, you do not need to deploy your own resource server implementation when using OpenAM as an OAuth 2.0 client. Instead, use policy agents or OpenIG to protect resources.

To configure OpenAM as an OAuth 2.0 client, see *Hints For the OAuth 2.0 Authentication Module*.

### 13.1.3. Using Your Own Client & Resource Server

OpenAM returns bearer tokens as described in RFC 6750, *The OAuth 2.0 Authorization Framework: Bearer Token Usage*. Notice in the following example JSON response to an access token request that OpenAM returns a refresh token with the access token. The client can use the refresh token to get a new access token as described in RFC 6749.

```

{
  "expires_in": 599,
  "token_type": "Bearer",
  "refresh_token": "f6dcf133-f00b-4943-a8d4-ee939fc1bf29",
  "access_token": "f9063e26-3a29-41ec-86de-1d0d68aa85e9"
}

```

In addition to implementing your client, the resource server must also implement the logic for handling access tokens. The resource server can use the `/oauth2/tokeninfo` endpoint to determine whether the access token is still valid, and to retrieve the scopes associated with the access token.

The default OpenAM implementation of OAuth 2.0 scopes assumes that the space-separated (%20 when URL encoded) list of scopes in an access token request correspond to names of attributes in the resource owner's profile.

To take a concrete example, consider an access token request where `scope=mail%20cn` and where the resource owner is the default OpenAM demo user. (The demo user has no email address by default, but you can add one, such as `demo@example.com` to the demo user's profile.) When the resource server performs an HTTP GET on the token information endpoint, `/oauth2/tokeninfo?access_token=token-id`, OpenAM populates the `mail` and `cn` scopes with the email address (`demo@example.com`) and common name (`demo`) from the demo user's profile. The result is a something like the following token information response.

```
{
  "mail": "demo@example.com",
  "scope": [
    "mail",
    "cn"
  ],
  "cn": "demo",
  "realm": "/",
  "token_type": "Bearer",
  "expires_in": 577,
  "access_token": "f9063e26-3a29-41ec-86de-1d0d68aa85e9"
}
```

OpenAM is designed to allow you to plug in your own scopes implementation if the default implementation does not do what your deployment requires. See *Customizing OAuth 2.0 Scope Handling* in the *Developer's Guide* for an example.

## 13.2. Configuring the OAuth 2.0 Authorization Service

You configure the OAuth 2.0 authorization service for a particular realm, starting from the Common Tasks page of the OpenAM console. This process also protects the authorization endpoint using a standard policy.

### *Procedure 13.1. To Set Up the OAuth 2.0 Authorization Service*

Follow these steps.

1. In the OpenAM console, select Common Tasks > Configure OAuth2.
2. On the Configure OAuth2 page, enter the Realm for the authorization service.
3. (Optional) If necessary, adjust the lifetimes for authorization codes (10 minutes is the recommended setting in RFC 6749), access tokens, and refresh tokens.

4. (Optional) Select Issue Refresh Tokens unless you do not want the authorization service to supply a refresh token when returning an access token.
5. (Optional) If you want to use the default scope implementation, whereby scopes are taken to be resource owner profile attribute names, then keep the default setting.

If you have a custom scope implementation, put it on the OpenAM classpath, and provide the class name as Scope Implementation Class.

6. Click Create to complete the process.

In addition to setting up an OAuth 2.0 authorization server for the realm, OpenAM sets up a policy to protect the authorization endpoint. The policy appears in the list of policies for the realm. Its name is `OAuth2ProviderPolicy`.

7. (Optional) If your provider has plugins for custom response types, add the custom response types and the corresponding plugin class names to the list of Response Type Plugins.
8. (Optional) If resource owners log in with a user name that is not their user ID, but instead their mail address or some other profile attribute, then add the profile attribute name to the list that can be used for authentication.

To make the change, browse to Access Control > *Realm Name* > Services > OAuth2 Provider, add the profile attributes to the list titled User Profile Attribute(s) the Resource Owner is Authenticated On.

9. Set a multi-valued profile attribute where OpenAM can store a resource owner's decisions to authorize clients without further interaction in the Shared Consent Attribute Name field.

If no profile attribute is available for this purpose, you can add an attribute as described in *Customizing Profile Attributes* in the *Developer's Guide* in the *Developer's Guide*.

10. Save your changes.

You can further adjust the authorization server configuration after you create it in the OpenAM console under Access Control > *Realm Name* > Services > OAuth2 Provider.

You can adjust global defaults in the OpenAM console under Configuration > Global > OAuth2 Provider.

## 13.3. Registering OAuth 2.0 Clients With the Authorization Service

You register an OAuth 2.0 client with the OpenAM OAuth 2.0 authorization service by creating and configuring an OAuth 2.0 Client agent profile.

At minimum you must have the client identifier and client password in order to register your OAuth 2.0 client.

### Procedure 13.2. To Create an OAuth 2.0 Client Agent Profile

- Use either of these two facilities.
  - In the OpenAM console, access the client registration endpoint at `/oauth2/registerClient.jsp`.

The full URL depends on where you deployed OpenAM. For example, `https://openam.example.com:8443/openam/oauth2/registerClient.jsp`.

The Register a Client page lets you quickly create and configure an OAuth 2.0 client in a simple web page without inline help.

- In the OpenAM console under Access Control > *Realm Name* > Agents > OAuth 2.0 Client > Agent, click New, then provide the client identifier and client password, and finally click Create to create the profile.

This page requires that you perform additional configuration separately.

### Procedure 13.3. To Configure an OAuth 2.0 Client Agent Profile

After initially registering or creating a client agent profile as necessary.

1. In the OpenAM console, browse to Access Control > *Realm Name* > Agents > OAuth 2.0 Client > Agent > *Client Name* to open the Edit *Client Name* page.
2. Adjust the configuration as needed using the inline help for hints, and also the documentation section *Configuring OAuth 2.0 & OpenID Connect 1.0 Clients*.

Examine the client type option. An important decision to make at this point is whether your client is a confidential client or a public client. This depends on whether your client can keep its credentials confidential, or whether its credentials can be exposed to the resource owner or other parties. If your client is a web-based application running on a server, such as the OpenAM OAuth 2.0 client, then you can keep its credentials confidential. If your client is a user-agent based client, such as a JavaScript client running in a browser, or a native application installed on a device used by the resource owner, then yours is a public client.

3. When finished, Save your work.

## 13.4. Managing OAuth 2.0 Tokens

OpenAM exposes a RESTful API that lets administrators read, list, and delete OAuth 2.0 tokens. OAuth 2.0 clients can also manage their own tokens. See the *Developer's Guide* section on the *OAuth 2.0 Token Administration Endpoint* in the *Developer's Guide* for details.

## 13.5. Configuring OpenAM as Authorization Server & Client

This section takes a high-level look at how to set up OpenAM both as an OAuth 2.0 authorization server and also as a client in order to protect resources by using a policy agent. The high-level steps are as follows.

1. Set up a policy agent or OpenIG to protect your web resources, including the policy used to protect the resources.  
  
Make sure you can access the resources using an authentication module that you already know before working with OAuth 2.0.
2. Configure OpenAM's OAuth 2.0 authorization service as described in Section 13.2, "Configuring the OAuth 2.0 Authorization Service".
3. Configure an OpenAM OAuth 2.0 authentication module instance using the *Hints For the OAuth 2.0 Authentication Module* with the OpenAM authorization service endpoints.
4. Register the OAuth 2.0 authentication module as an OAuth 2.0 confidential client as described in Section 13.3, "Registering OAuth 2.0 Clients With the Authorization Service".
5. Logout and access the protected resources to see the process in action.

### Example 13.1. Web Site Protected With OAuth 2.0

This example pulls everything together (except security considerations), using OpenAM as an OAuth 2.0 authorization server, OAuth 2.0 client, and "resource server" by protecting web resources with a policy agent.

This example uses two hosts, `oauth2.example.com` where OpenAM is deployed at `http://oauth2.example.com:8080/openam`, and `www.example.com` where the resources to protect are deployed in Apache Tomcat at `http://www.example.com:8080/examples`.

1. Set up an OpenAM policy agent and policy in the top-level realm, `/`, to protect resources. So far this is standard OpenAM, unrelated to OAuth 2.0.

See the *OpenAM Web Policy Agent 3.3.0 Installation Guide* or *OpenAM Java EE Policy Agent 3.3.0 Installation Guide* for instructions on installing a policy agent. This example relies on the Apache Tomcat Java EE policy agent, configured to protect resources in Apache Tomcat at `http://www.example.com:8080/`.

The policies for this example protect the Apache Tomcat examples under `http://www.example.com:8080/examples/`, allowing GET and POST operations by all authenticated users. For more information on creating policies, see *Configuring Policies*.

After setting up the policy agent and the policy, you can make sure everything is working by attempting to access a protected resource, in this case `http://www.example.com:8080/examples/`. The

policy agent should redirect you to OpenAM to authenticate with the default authentication module, where you can login as user `demo` password `changeit`. After successful authentication, OpenAM redirects your browser back to the protected resource and the policy agent lets you get the protected resource, in this case the Tomcat examples top page.

### Apache Tomcat Examples

- [Servlets examples](#)
- [JSP Examples](#)
- [WebSocket Examples](#)

2. Configure OpenAM's OAuth 2.0 authorization service as described in Section 13.2, "Configuring the OAuth 2.0 Authorization Service".

The authorization endpoint to protect in this example is at <http://oauth2.example.com:8080/openam/oauth2/authorize>.

3. Configure an OpenAM OAuth 2.0 authentication module instance for the top-level realm.

Under Access Control > / (Top-Level Realm) > Authentication > Module Instances, click New. Name the module `OAuth2`, and select the OAuth 2.0 type, then click OK to save your work.

Then click Authentication > Module Instances > OAuth2 to open the OAuth 2.0 client configuration page. This page offers numerous options. The key settings for this example are the following.

#### Client Id

This is the client identifier used to register your client with OpenAM's authorization server, and then used when your client must authenticate to OpenAM.

Set this to `myClientID` for this example.

#### Client Secret

This is the client password used to register your client with OpenAM's authorization server, and then used when your client must authenticate to OpenAM.

Set this to `password` for this example. Make sure you use strong passwords when you actually deploy OAuth 2.0.

#### Authentication Endpoint URL

In this example, <http://oauth2.example.com:8080/openam/oauth2/authorize>.

This OpenAM endpoint can take additional parameters. In particular you must specify the realm if the OpenAM OAuth 2.0 provider is configured for a subrealm rather than / (Top-Level Realm).

For example, if the OAuth 2.0 provider is configured for the realm `/customers`, then use the following URL: <https://openam.example.com:8443/openam/oauth2/authorize?realm=/customers>

The `/oauth2/authorize` endpoint can also take `module` and `service` parameters. Use either as described in *Authenticating To OpenAM*, where `module` specifies the authentication module instance to use or `service` specifies the authentication chain to use when authenticating the resource owner.

### Access Token Endpoint URL

In this example, `http://oauth2.example.com:8080/openam/oauth2/access_token`.

This OpenAM endpoint can take additional parameters. In particular you must specify the realm if the OpenAM OAuth 2.0 provider is configured for a subrealm rather than / (Top-Level Realm).

For example, if the OAuth 2.0 provider is configured for the realm `/customers`, then use the following URL: `https://openam.example.com:8443/openam/oauth2/access_token?realm=/customers`

### User Profile Service URL

In this example, `http://oauth2.example.com:8080/openam/oauth2/tokeninfo`.

### Scope

In this example, `cn`.

The demo user has common name `demo` by default, so by setting this to `cn|Read your user name`, OpenAM can get the value of the attribute without the need to create additional subjects, or to update existing subjects. The description, `Read your user name`, is shown to the resource owner in the consent page.

### OAuth2 Access Token Profile Service Parameter name

Identifies the parameter that contains the access token value, which in this example is `access_token`.

### Proxy URL

The client redirect URL, which in this example is `http://oauth2.example.com:8080/openam/oauth2c/OAuthProxy.jsp`.

### Attribute Mapper

In this example, `org.forgerock.openam.authentication.modules.oauth2.DefaultAccountMapper`.

### Account Mapper Configuration

In this example, `cn=cn`.

## Create account if it does not exist

In this example, disable this functionality.

OpenAM can create local accounts based on the account information returned by the authorization server. For this example, map all users to the anonymous account to keep it simple.

## Map to anonymous user

In this example, enable this functionality.

4. Register the OAuth 2.0 authentication module as an OAuth 2.0 confidential client as described in Section 13.3, "Registering OAuth 2.0 Clients With the Authorization Service".

Under Access Control > / (Top-Level Realm) > Agents > OAuth 2.0 Client > Agents > `myClientID`, adjust the following settings.

### Client type

In this example, `confidential`. OpenAM protects its credentials as an OAuth 2.0 client.

### Redirection URIs

In this example, `http://oauth2.example.com:8080/openam/oauth2c/OAuthProxy.jsp`.

### Scopes

In this example, `cn`.

5. Before you try it out, you must make the following additional change to the configuration.

Your OpenAM OAuth 2.0 client authentication module is not part of the default chain, and therefore OpenAM does not call it unless you specifically request the OAuth 2.0 client authentication module.

To cause the policy agent to request your OAuth 2.0 client authentication module explicitly, browse in OpenAM console to your *policy agent profile configuration*, in this case Access Control > / (Top-Level Realm) > Agents > J2EE > Agents > `Tomcat` > OpenAM Services > OpenAM Login URL, and add `http://oauth2.example.com:8080/openam/UI/Login?module=OAuth2`, moving it to the top of the list.

Save your work.

This ensures that the policy agent directs the resource owner to OpenAM with the instruction to authenticate using the `OAuth2` authentication module.

6. Try it out.

First make sure you are logged out of OpenAM, for example by browsing to the logout URL, in this case <http://oauth2.example.com:8080/openam/UI/Logout>.

Next attempt to access the protected resource, in this case <http://www.example.com:8080/examples/>.

If everything is set up properly, the policy agent redirects your browser to the login page of OpenAM with `module=OAuth2` among other query string parameters. After you authenticate, for example as user `demo`, password `changeit`, OpenAM presents you with an authorization decision page.

## OAuth authorization page

### Application requesting scope

Example.com intranet:

**The following private info is requested**

**Read your user name**

Save Consent:  Allow  Deny

When you click Allow, the authorization service creates an SSO session, and redirects the client back to the resource, thus allowing the client to access the protected resource. If you configured an attribute on which to store the saved consent decision, and you choose to save the consent decision for this authorization, then OpenAM can use that saved decision to avoid prompting you for authorization next time the client accesses the resource, but only ensure that you have authenticated and have a valid session.

### Apache Tomcat Examples

- [Servlets examples](#)
- [JSP Examples](#)
- [WebSocket Examples](#)

## 13.6. Configuring OpenAM as Both SAML 2.0 Service Provider & OAuth 2.0 Authorization Server

As described in Section 13.1.1.5, "SAML 2.0 Bearer Assertion Profiles", OpenAM as OAuth 2.0 authorization server can handle the profile where a SAML 2.0 assertion borne by the client functions as an authorization grant to get an access token. This lets a client get an access token when a resource owner completes SAML 2.0 Web Single Sign-On.

You can configure OpenAM as both SAML 2.0 service provider and OAuth 2.0 authorization server, using an built-in adapter class to POST assertions returned to the service provider to the access

token endpoint of the authorization server. This allows clients to send a resource owner to the identity provider for SAML 2.0 web SSO, get an assertion at the service provider, and retrieve an access token from the authorization server. In other words, once this scenario is configured, the client must only direct the resource owner to start web SSO as described in *Using SAML 2.0 Single Sign-On & Single Logout*, and then retrieve the access token on success or handle the error condition on failure.

#### Procedure 13.4. To Get an Access Token From SAML 2.0 Web SSO

For this scenario to work, the following conditions must be met.

- The client must make the resource owner understand that by authenticating to the SAML 2.0 identity provider the resource owner grants the client access to the protected resources. OpenAM does not present the resource owner with an authorization decision.
- The SAML 2.0 identity provider issuing the assertion must sign the assertion, and must correctly handle the name ID for the subject.
- OpenAM as relying party must request that assertions are signed, must verify the signatures on assertions, must correctly handle name IDs from the issuer, and must use the built-in `org.forgerock.restlet.ext.oauth2.flow.OAuth2Saml2GrantSPAdapter` adapter class in the service provider configuration to POST assertions to the OAuth 2.0 authorization service.
- The OAuth 2.0 authorization service and SAML 2.0 service provider must be configured together on the same OpenAM server.
- An OAuth 2.0 client configuration on OpenAM with the same name as the service provider entity ID must be set up on OpenAM.
- The OAuth 2.0 client initiating the process must be able to consume the access token and to handle errors if necessary.

Follow these steps. The test configuration hints in this procedure let you prepare configuration to test with the demo user created in OpenAM by default.

1. Make sure the SAML 2.0 identity provider signs assertions and that name IDs are correctly configured to map resource owner accounts.

When configuring OpenAM as a hosted identity provider follow these steps.

- a. Make sure the Signing Key is properly configured on setup.

For a test configuration, select the `test` certificate shown in the Common Tasks > Create Hosted Service Provider wizard.

- b. Make sure name IDs are properly configured.

For a test configuration, in the OpenAM console under Federation > Entity Providers > *IDP name* > NameID Value Map, add `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified=cn` and then Save your work.

For more detail on configuring OpenAM as a SAML 2.0 identity provider, see *Configuring Identity Providers*.

## 2. Configure OpenAM as service provider.

- a. Set up a hosted service provider in OpenAM console under Common Tasks > Create Hosted Service Provider, keeping track of the name, such as <https://www.sp.example:8443/openam>, and selecting Use default attribute mapping from Identity Provider.

For details on configuring OpenAM as a SAML 2.0 service provider, see *Configuring Service Providers*.

- b. Under Federation > Entity Providers > *SP name* > Assertion Content > Request/Response Signing, check Assertions Signed.
- c. For a test configuration, in Federation > Entity Providers > *SP name* > Assertion Content > NameID Format List, remove all but `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified`, and then Save your work.
- d. In Federation > Entity Providers > *SP name* > Assertion Processing > Adapter, add `org.forgerock.restlet.ext.oauth2.flow.OAuth2Saml2GrantSPAdapter`, and then Save your work.

This is the adapter class that POSTs the SAML 2.0 assertion to the OAuth 2.0 access token endpoint.

- e. Use the wizard under Common Tasks > Register Remote Identity Provider to import the identity provider metadata.

## 3. Make sure the identity provider imports the metadata for your service provider.

If your service provider is at <https://www.sp.example:8443/openam>, then the metadata can be accessed at <https://www.sp.example:8443/openam/saml2/jsp/exportmetadata.jsp>.

4. On the service provider OpenAM server, set up the OAuth 2.0 authorization server as described in Section 13.2, "Configuring the OAuth 2.0 Authorization Service".

For a test configuration, set the realm to `/`, and accept the defaults.

5. On the service provider and authorization server OpenAM server, set up an OAuth 2.0 client profile with the same name as the service provider under Access Control > *realm* > Agents > OAuth 2.0 Client > New...

For example, if the service provider name is <https://www.sp.example:8443/openam>, then that is also the name of the OAuth 2.0 client profile.

You can make additional changes to the client profile if necessary. See Section 13.3, "Registering OAuth 2.0 Clients With the Authorization Service" for details.

## 6. Test your configuration.

- a. Logout of all OpenAM servers.
- b. Initiate SAML 2.0 Web SSO.

For example, if your identity provider is at <https://www.idp.example:8443/openam> with meta alias `/idp` and your service provider is at <https://www.sp.example:8443/openam>, then browse to the following URL (without line breaks or spaces).

```
http://www.idp.example:8443/openam/saml2/jsp/idpSSOInit.jsp
?metaAlias=/idp&spEntityID=http://www.sp.example:8443/openam
```

For other configurations, see *Using SAML 2.0 Single Sign-On & Single Logout*.

- c. Login to the identity provider.

For OpenAM, login with user name `demo` and password `changeit`.

- d. Login to the service provider.

For OpenAM, login with user name `demo` and password `changeit`.

- e. See the resulting access token on successful login.

The result looks something like this, all on one line.

```
{
  "expires_in": 59,
  "token_type": "Bearer",
  "access_token": "f0f731e0-6013-47e3-9c07-da598157a85f"
}
```

## 13.7. Security Considerations

OAuth 2.0 messages involve credentials and access tokens that allow the bearer to retrieve protected resources. Therefore, do not let an attacker capture requests or responses. Protect the messages going across the network.

RFC 6749 includes a number of *Security Considerations*, and also requires Transport Layer Security (TLS) to protect sensitive messages. Make sure you read the section covering *Security Considerations*, and that you can implement them in your deployment.

Also, especially when deploying a mix of other clients and resource servers, take into account the points covered in the Internet-Draft, *OAuth 2.0 Threat Model and Security Considerations*, before putting your service into production.

## Chapter 14

# Managing OpenID Connect 1.0 Authorization

This chapter covers OpenAM support for OpenID Connect 1.0. OpenID Connect 1.0 extends OAuth 2.0 so the client can verify claims about the identity of the end user, get profile information about the end user, and log the user out at the end of the OpenAM session. OpenID Connect also makes it possible to discover the provider for an end user, and to register client applications dynamically. OpenID connect services are built on OAuth 2.0, JSON Web Token (JWT), WebFinger and Well-Known URIs.

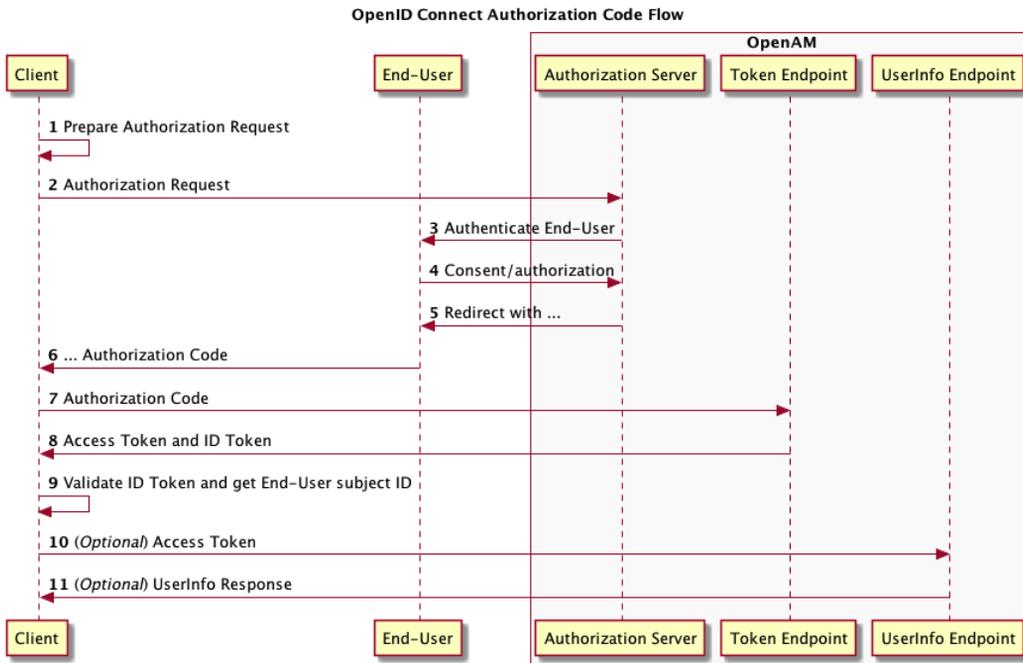
## 14.1. About OpenID Connect 1.0 Support in OpenAM

In its role as OpenID Provider, OpenAM lets OpenID Connect clients discover its capabilities, handles both dynamic and static registration of OpenID Connect clients, responds to client requests with authorization codes, access tokens, and user information according to the Basic and Implicit Client Profiles of OpenID Connect, and manages sessions.

This section describes how OpenAM fits into the OpenID Connect picture in terms of the roles that it plays in basic and implicit client profiles, provider discovery, client registration, and session management.

### 14.1.1. OpenID Connect Basic Client Profile

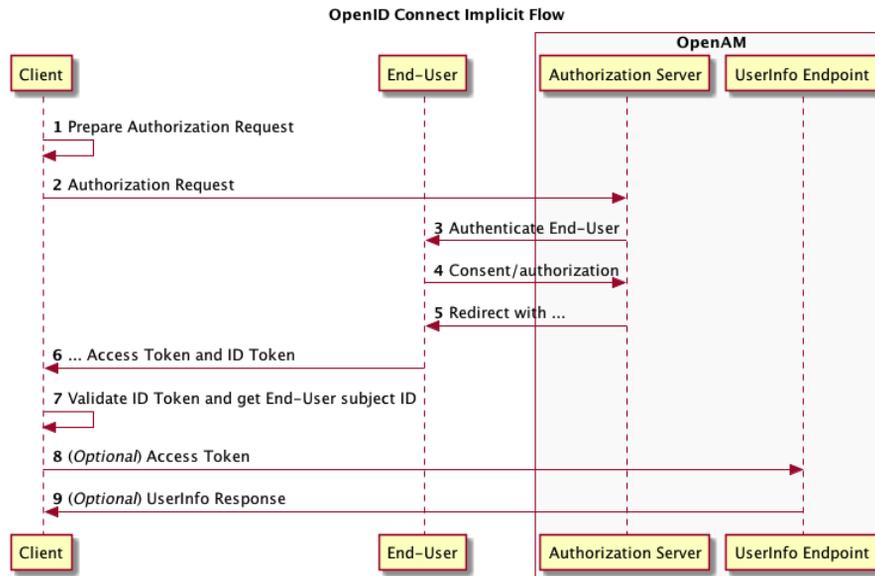
The OpenID Connect Basic Client Profile specifies how the client application interacts with the provider, in this case OpenAM, based on use of the OAuth 2.0 authorization grant. The following sequence diagram shows successful processing from the authorization request, through grant of the authorization code, access token, and ID token, and optional use of the access token to get information about the end user.



In addition to what OAuth 2.0 specifies, OpenID Connect uses an ID token so the client can validate claims about the end user. It also defines how to get user information such as profile, email, address, and phone details from the UserInfo endpoint with a valid access token.

### 14.1.2. OpenID Connect Implicit Client Profile

The OpenID Connect Implicit Client Profile specifies how the client application interacts with the provider, in this case OpenAM, based on use of the OAuth 2.0 implicit grant. The following sequence diagram shows successful processing from the authorization request, through grant of the access and ID tokens, and optional use of the access token to get information about the end user.



As for the Basic Client Profile, the Implicit Client Profile specifies an ID token so the client can validate claims about the end user. It also defines how to get user information such as profile, email, address, and phone details from the UserInfo endpoint with a valid access token.

### 14.1.3. OpenID Connect Discovery

OpenID Connect defines how a client application can discover the OpenID connect provider and corresponding OpenID Connect configuration for an end user. The discovery mechanism relies on WebFinger to get the information based on the end user's identifier. The server returns the information in JSON Resource Descriptor (JRD) format.

### 14.1.4. OpenID Connect Client Registration

OpenID Connect clients can register with OpenAM as a provider both statically, as for other OAuth 2.0 clients, and also dynamically as specified by OpenID Connect. To allow dynamic registration, you register an initial client that other clients can use to get access tokens for registration.

### 14.1.5. OpenID Connect Session Management

OpenID Connect lets the client track whether the end user is logged in at the provider, and also initiate end user logout at the provider. The specification has the client application monitor session state using an invisible iframe and communicate status using the HTML 5 postMessage API.

## 14.2. Configuring OpenAM As an OpenID Connect Provider

You can configure OpenAM's OAuth 2.0 authorization service to double as an OpenID Connect 1.0 Provider. To do so, make sure that the Response Type Plugins list includes at least the default plugin classes.

See *Configuring the OAuth 2.0 Authorization Service* for detailed instructions on configuring the service.

## 14.3. Configuring OpenAM For OpenID Connect Discovery

In order to allow clients to discover the provider for an end user, OpenAM supports OpenID Connect Discovery 1.0. In addition to discovering the provider for an end user, the client can also request the OpenID Connect Provider configuration.

OpenAM as OpenID Connect Provider exposes two endpoints for discovery:

```
/.well-known/webfinger  
/.well-known/openid-configuration
```

A client needs to be able to discover the provider for an end user. In this case you should consider redirecting requests to URIs at the server root, such as <http://www.example.com/.well-known/webfinger> and <http://www.example.com/.well-known/openid-configuration>, to these Well-Known URIs in OpenAM's space.

Discovery relies on [WebFinger](#), a protocol to discover information about people and other entities using standard HTTP methods. WebFinger uses [Well-Known URIs](#), which defines the path prefix `/.well-known/` for the URLs defined by OpenID Connect Discovery.

Unless you deploy OpenAM in the root context of a container listening on port 80 on the primary host for your domain, clients need to find the right `host:port/deployment-uri` combination to locate the well-known endpoints. Therefore you must manage the redirection to OpenAM. If you are using WebFinger for something else than OpenID Connect Discovery, then you probably also need proxy logic to route the requests.

To retrieve the provider for an end user, the client needs the following.

### host

The server where the client can access the WebFinger service.

Notice that this is a host name rather than a URL to the endpoint, which is why you might need to redirect clients appropriately as described above.

### resource

Identifies the end user that is the subject of the request.

The client must percent-encode the resource value when using it in the query string of the request, so when using the "acct" URI scheme and the resource is `acct:user@example.com`, then the value to use is `acct%3Auser%40example.com`.

## rel

URI identifying the type of service whose location is requested.

In this case `http://openid.net/specs/connect/1.0/issuer`, which is `http%3A%2F%2Fopenid.net%2Fspecs%2Fconnect%2F1.0%2Fissuer`.

Ignoring the question of redirection, you can test the endpoint for the demo user account (output lines folded to make them easier to read).

```
$ curl "https://openam.example.com:8443/openam/.well-known/webfinger
?resource=acct%3Ademo%40example.com
&rel=http%3A%2F%2Fopenid.net%2Fspecs%2Fconnect%2F1.0%2Fissuer"
{
  "subject": "acct:demo@example.com",
  "links": [
    {
      "rel": "http://openid.net/specs/connect/1.0/issuer",
      "href": "https://openam.example.com:8443/openam"
    }
  ]
}
```

This shows that the OpenID Connect Provider for the OpenAM demo user is indeed the OpenAM server.

The client can also discover the provider configuration. Ignoring the question of redirection, you can test this (output lines folded to make them easier to read).

```
{
  "response_types_supported": [
    "id_token|org.forgerock.restlet.ext.oauth2.flow.responseTypes.IDTokenResponseType",
    "token|org.forgerock.restlet.ext.oauth2.flow.responseTypes.TokenResponseType",
    "code|org.forgerock.restlet.ext.oauth2.flow.responseTypes.CodeResponseType"
  ],
  "registration_endpoint": "https://openam.example.com:8443/openam/oauth2/connect/register",
  "token_endpoint": "https://openam.example.com:8443/openam/oauth2/access_token",
  "end_session_endpoint": "https://openam.example.com:8443/openam/oauth2/connect/endSession",
  "version": "3.0",
  "userinfo_endpoint": "https://openam.example.com:8443/openam/oauth2/userinfo",
  "subject_types_supported": [
    "pairwise",
    "public"
  ],
  "issuer": "https://openam.example.com:8443/openam",
  "jwks_uri": "",
  "id_token_siging_alg_values_supported": [
    "HmacSHA256",
    "HmacSHA512",
    "HmacSHA384"
  ]
}
```

```
    ],  
    "check_session_iframe": "https://openam.example.com:8443/openam/oauth2/connect/checkSession",  
    "claims_supported": [  
        "phone",  
        "email",  
        "address",  
        "openid",  
        "profile"  
    ],  
    "authorization_endpoint": "https://openam.example.com:8443/openam/oauth2/authorize"  
}
```

## 14.4. Registering OpenID Connect Clients

OpenID Connect Clients can register with OpenAM both statically through OpenAM console for example, and also dynamically using OpenID Connect 1.0 Dynamic Registration.

### *Procedure 14.1. To Register a Client With OpenAM Console*

Registering a client by using the OpenAM console consists of first creating an OAuth 2.0 Client agent profile, and then editing the profile to indicate the client settings pertinent to OpenID Connect 1.0.

1. In the OpenAM console under Access Control > *Realm Name* > Agents > OAuth 2.0 Client > Agent, click New..., then provide the client identifier and client password, and finally click Create to create the profile.
2. Follow the hints in the section, *Configuring OAuth 2.0 & OpenID Connect 1.0 Clients* to edit the profile to match the client configuration.

In order to read and edit the client configuration dynamically later without using OpenAM console, be sure to set an access token in the field titled, "The access token used to update the client."

### *Procedure 14.2. To Register a Client Dynamically*

For dynamic registration you need the client configuration metadata, and an access token to write the configuration to OpenAM by HTTP POST. To obtain the access token, register an initial client statically after creating the provider, as described in Procedure 14.1, "To Register a Client With OpenAM Console". Other clients can then use that client to obtain the access token needed to perform dynamic registration.

On successful registration, OpenAM responds with information including an access token to allow the client subsequently to read and edit its profile.

1. Register an initial OAuth 2.0 client statically with a client ID such as `masterClient` and client secret such as `password`.
2. Obtain an access token using the client you registered.

For example, if you created the client as described in the previous step, and OpenAM administrator `amadmin` has password `password`, you can use the OAuth 2.0 resource owner password grant as in the following example.

```
$ curl
--request POST
--user "masterClient:password"
--data "grant_type=password&username=amadmin&password=password"
https://openam.example.com:8443/openam/oauth2/access_token
{
  "expires_in": 59,
  "token_type": "Bearer",
  "refresh_token": "26938cd0-6870-4e31-ade9-df31afc37ee1",
  "access_token": "515d6551-4512-4279-98b6-c0ef3f03a722"
}
```

3. HTTP POST the client registration profile to the `/oauth2/connect/register` endpoint, using bearer token authorization with the access token you obtained from OpenAM.

For an example written in JavaScript, see the registration page in the examples available online. Successful registration shows a response that includes the client ID and client secret.

```
{
  "issued_at": 1376916775,
  "expires_at": 0,
  "client_secret": "5a1f50db-ed93-4a9b-bf37-c1cb3c74ca16",
  "com.forgerock.openam.oauth2provider.redirectionURIs": [
    "https://openam.example.com:8443/openid/cb-basic.html",
    "https://openam.example.com:8443/openid/cb-implicit.html"
  ],
  "client_id": "c1e5c60a-9fd4-4780-a283-ec89580b6cc1",
  "registration_client_uri":
    "https://openam.example.com:8443/openam/oauth2/connect/register
    ?client_id=c1e5c60a-9fd4-4780-a283-ec89580b6cc1"
}
```

## 14.5. Managing User Sessions

OpenID Connect Session Management 1.0 allows the client to manage OpenID Connect sessions, making it possible to know when the end user should be logged out.

As described in the [OpenID Connect Session Management 1.0 specification](#), OpenAM's OpenID Connect provider exposes both a "check\_session\_iframe" URL that allows the client to receive notifications when the end user's session state changes at the provider, and also an "end\_session\_endpoint" URL to which to redirect an end user for logout.

When registering your client that uses session management, you set the OAuth 2.0 client agent profile properties Post Logout Redirect URI and Client Session URI, described in [Configuring OAuth 2.0 & OpenID Connect 1.0 Clients](#). The Post Logout Redirect URI is used to redirect the end user user-agent after logout. The Client Session URI is the client URI where OpenAM sends notifications when the end user's session state changes.

## 14.6. Client Examples

OpenID Connect Basic and Implicit Client Profiles define how clients interact with the provider to obtain end user authorization and profile information. Although you can run the simple example clients mentioned in this section without setting up Transport Layer Security, do not deploy clients in production without securing the transport.

Code for the client examples shown here is available online. Clone the example project to deploy it in the same web container as OpenAM. Edit the configuration at the outset of the .js files in the project, register a corresponding profile for the example client as described in Section 14.4, "Registering OpenID Connect Clients", and browse the deployment URL to see the initial page.

### OpenID Connect Client Profiles



[OpenID Connect 1.0](#) defines two client profiles.

#### *Basic Client Profile*

The Basic Client Profile is designed for web-based relying parties that use the OAuth 2.0 Authorization Code grant type, such as server-side clients that can protect their client credentials.

Try the [Basic Client Profile](#).

#### *Implicit Client Profile*

The Implicit Client Profile is designed for relying parties that use the OAuth 2.0 Implicit grant type, such as browser-based clients written in JavaScript.

Try the [Implicit Client Profile](#).

The examples provided here are both written in JavaScript. Neither aims to protect anything, but instead to show you the steps that each kind of client follows, and the responses from OpenAM as OpenID Connect Provider.

### *Example 14.1. Basic Client Profile Example*

OpenID Connect Basic Client Profile 1.0 is designed for web-based relying parties that use the OAuth 2.0 Authorization Code grant type. This grant type makes it possible for the client to get the access code by using the authorization code directly, without passing through the end user's browser. To protect its client secret (password), part of the client must run on the server.

In the example, the Basic Client Profile Start Page describes the prerequisite configuration, which must be part of the client profile stored in the OpenAM realm where you set up the OpenID Provider. In OpenAM console, check that the OAuth 2.0 agent profile matches the settings described.

## Basic Client Profile Start Page



Try OpenAM as an OpenID Connect provider using the [Basic Client Profile](#).

OpenID Connect Basic Client Profile 1.0 is designed for web-based relying parties that use the OAuth 2.0 Authorization Code grant type. This grant type makes it possible for the client to get the access code by using the authorization code directly, without passing through the end user's browser. To protect its client secret (password), part of the client must run on the server.

Note: This example is not designed to protect the client secret, but instead to show the results at each step in the process.

---

### Prerequisite Configuration

OpenAM should be running and configured as an OpenID Connect Provider in the same container as this application.

Current settings in `basic.js`:

```
OpenAM URI      /openam
client_id       myClientID
client_secret   password
redirect_uri    http://openam.example.com:8080/openid/cb-basic.html
```

In OpenAM, create an OAuth 2.0 agent using the `client_id`, `client_secret`, and `redirect_uri`, and then edit the configuration to add the scopes "openid" and "profile".

After you have configured everything, log out of OpenAM. Then click the link to start the authorization process.

[Start authorization](#)

Logout of OpenAM, and click the link at the bottom of the page to request authorization. The link sends an HTTP GET request asking for `openid profile` scopes to the OpenID Connect provider authorization URI.

If everything is configured correctly, OpenAM's OpenID Connect provider has you authenticate as an end user, such as the demo user with username `demo` and password `changeit`, and grant (Allow) the client access to your profile.

If you successfully authenticate and allow the example client access to your profile, OpenAM returns an authorization code to the example client. The example client then uses the authorization code to request an access token and an ID token. It shows the response to that request, and also decodes the ID token to show the content and to perform some validation. Finally it uses the access token to request information about the end user who authenticated, and displays the result.

## Basic Client Profile Response Page



### Authorization Code

```
db6161f5-f1e8-4d89-aadc-6dc0d391e0d4
```

### Token Response

```
{
  "expires_in": 59,
  "token_type": "Bearer",
  "refresh_token": "86fd086c-312f-4ce7-abb6-399792e269b7",
  "id_token": "eyJYXNzIjogIkhTMjU2IiwgInR5cCI6ICJKV1QiIH0=.eyJYXRoIjogMTMzMjY4OTE4NCwgImF1ZCI6ICJteUNsaWVudE1EiiwgImF6cCI6ICJteUNsaWVudE1EiiwgImV4cCI6IDEzNzI0ODkyNDQsICJpYXQiOiAxMzcyNjg5MTg0LCAiaXNzIjogImh0dHA6Ly9vcGVuYW0uZXhhbXBsZS5jb206ODAwMzY0Ym90LCAicmVhbnQ0IiAiLyIsICJzdWIiOiAiZGVtbyIsICJ0b2t1bk5hbWUiOiAiaWRfdG9rZW4iLCAidG9rZW5UeXB1IjogIkpXVFRva2VuIiB9.OaNLbJpfy51fJZgwg6pJnBkNWA/yh0/smAsSF0b6Lw=",
  "access_token": "59a259df-77da-485c-8e51-59a06038f926"
}
```

### Decoded ID Token Content

```
{
  "ath": 1372689184,
  "aud": "myClientID",
  "azp": "myClientID",
  "exp": 1372689244,
  "iat": 1372689184,
  "iss": "http://openam.example.com:8080/openam",
  "realm": "/",
  "sub": "demo",
  "tokenName": "id_token",
  "tokenType": "JWTToken"
}
```

### End User Info

```
{
  "sub": "demo",
  "name": "demo",
  "family_name": "demo"
}
```

[Try basic profile again](#) | [Try implicit profile](#) | [Start over](#)

Notice that in addition to the standard payload, the ID token indicates the end user's OpenAM realm, in this case `"realm": "/"`.

### Example 14.2. Implicit Client Profile Example

OpenID Connect Implicit Client Profile 1.0 is designed for relying parties that use the OAuth 2.0 Implicit grant type. This grant type is designed for clients implemented in a browser. Rather than protect a client secret, the client profile must register a protected redirect URI in advance with the OpenID Provider.

In the example, the Implicit Client Profile Start Page describes the prerequisite configuration, which must be part of the client profile stored in the OpenAM realm where you set up the OpenID Provider. In OpenAM console, check that the OAuth 2.0 agent profile matches

the settings described. If you have already configured the agent profile for the Basic Client Profile then you still need to add the redirect URI for the Implicit Client Profile.

## Implicit Client Profile Start Page



This example tries OpenAM as an OpenID Connect provider using the [Implicit Client Profile](#).

OpenID Connect Implicit Client Profile 1.0 is designed for relying parties that use the OAuth 2.0 Implicit grant type. This grant type is designed for clients implemented in a browser. Rather than protect a client secret, the client profile must register a protected redirect URI in advance with the OpenID Provider.

---

### Prerequisite Configuration

OpenAM should be running and configured as an OpenID Connect Provider in the same container as this application.

Current settings for this example:

```
OpenAM URI      /openam
client_id       myClientID
redirect_uri    http://openam.example.com:8080/openid/cb-implicit.html
```

In OpenAM, create an OAuth 2.0 agent using the `client_id`, then edit the configuration to add the `redirect_uri`, and scopes "openid" and "profile".

After you have configured everything, log out of OpenAM. Then click the link to start the authorization process.

[Start authorization](#)

Logout of OpenAM, and click the link at the bottom of the page to request authorization.

The link sends an HTTP GET request asking for `id_token` response types and `openid profile` scopes to the OpenID Connect provider authorization URI.

If everything is configured correctly, OpenAM's OpenID Connect provider has you authenticate as an end user, such as the demo user with username `demo` and password `changeit`, and grant (Allow) the client access to your profile.

If you successfully authenticate and allow the example client access to your profile, OpenAM returns the access token and ID token directly in the fragment (after #) of the redirect URI. The client does not get an authorization code. The client shows the response to the request, and also decodes the ID token to show the content and to perform some validation (though it does not check the ID token signature). Finally the client uses the access token to request information about the end user who authenticated, and displays the result.

## Implicit Client Profile Response Page



Note: No validation done for id\_token signatures.

### Response From Provider

```
{
  "id_token": "eyJhdHlwIjogIkpXVCJgfQ==.eyJhYXRoIjogMTM3MjY5MDE5MSwgImF1ZCI6ICJteUNsaWVudE1EiIiwgImF6cCI6ICJteUNsaWVudE1EiIiwgImV4cCI6IDEzNzI0OTYyNTYsICJpYXQiOiAxMzcyNjkwMTkxLCAiaXNzIjogImh0dHA6Ly9vcGVuYW0uZXhhbXBsZS5jb206ODA4MC9vcGVuYW0iLCAibm9uY2UiOiAiIiwuUzZfV3pBMk1qIiwgInJlYWxtIjogIi8iLCAic3ViIjogImRlbW8iLCAidG9rZW50YWllIiwgImlkX3Rva2VuIiwgInRva2VuVHlwZSI6ICJKV1RUb2t1biIgfQ==.",
  "access_token": "23202492-de24-4b3b-ae6c-03ee16cbf14",
  "expires_in": "59",
  "token_type": "Bearer",
  "state": "af0ifjsldkj"
}
```

### Decoded ID Token Content

```
{
  "ath": 1372690191,
  "aud": "myClientID",
  "azp": "myClientID",
  "exp": 1372690251,
  "iat": 1372690191,
  "iss": "http://openam.example.com:8080/openam",
  "nonce": "n-0S6_WzA2Mj",
  "realm": "/",
  "sub": "demo",
  "tokenName": "id_token",
  "tokenType": "JWTToken"
}
```

### Profile Information

```
{
  "sub": "demo",
  "name": "demo",
  "family_name": "demo"
}
```

[Try implicit profile again](#) | [Try basic profile](#) | [Start over](#)

As for the Basic Client Profile, the ID Token indicates the end user's OpenAM realm in addition to the standard information.

## 14.7. Security Considerations

As for other OAuth 2.0 applications, you must protect messages going across the network. OpenID Connect 1.0 requires Transport Layer Security (TLS). The chapter on *Managing Certificates* includes some discussion of protecting traffic in the container where OpenAM runs. Also see the documentation for your web application container.

Also take into account the points developed in the section on *Security Considerations* in the OpenID Connect Messages draft specification.

## Chapter 15

# Backing Up and Restoring OpenAM Configurations

This chapter shows how to backup and restore OpenAM configuration data. This chapter *does not cover backup and restore of user data*, which in a critical production system should be stored separately.

### Procedure 15.1. To Back Up OpenAM Configuration

OpenAM stores service configuration data in a directory. During normal production operations, you rely on directory replication to maintain multiple, current copies of OpenAM service configuration. For disaster recovery, however, you backup to and restore the service configuration from XML, using the **ssoadm** command.

1. Backup OpenAM service configuration using the **ssoadm** command.

```
$ ssoadm export-svc-cfg -u amadmin -e fZatIu680iqccJMXosSRyVjMswJIx+SA  
-f /tmp/pwd.txt -o ~/backup-`date -u +%F-%m-%S`.xml
```

```
Service Configuration was exported.
```

In this example, the secret key for encrypting the password in `-e fZatIu680iqccJMXosSRyVjMswJIx+SA` was taken from the Password Encryption Key field in the OpenAM console under Configuration > Servers and Sites > *Server Name* > Security.

2. Stop OpenAM.
3. Back up the instance file that points to the configuration directory.

This file is named after the instance location, such as `$HOME/.openamcfg/AMConfig_path_to_tomcat_webapps_openam_`, where `$HOME` is the home directory of the user running the web container where OpenAM is deployed.

4. Back up the files in the configuration directory.

The content of the file you backed up in the previous step is the path to the configuration directory, such as `$HOME/openam`.

5. Start OpenAM.

### Procedure 15.2. To Restore OpenAM Configuration

The following steps restore OpenAM configuration data from backup as described in Procedure 15.1, "To Back Up OpenAM Configuration".

**Tip**

If using the default OpenAM configuration data store, run **ssoadm embedded-status** to check the data store status to determine whether you must restore the configuration files including the embedded data store, or only the service configuration.

1. Deploy the OpenAM web application as you did for installation, but do not start OpenAM or configure it.

In a site configuration, perform this step on all servers.

2. Restore files in the configuration directory as necessary.

In a site configuration, perform this step on all servers.

3. Restore the bootstrap files as necessary.

In a site configuration, perform this step on all servers.

4. Start OpenAM.

In a site configuration, perform this step on all servers before proceeding.

5. Restore OpenAM service configuration using the **ssoadm** command.

```
$ ssoadm import-svc-cfg -u amadmin -e fZatIu680iqccJMXosSRyVjMswJIx+SA  
-f /tmp/pwd.txt -X ~/backup-2011-09-13-09-00.xml
```

```
Directory Service contains existing data. Do you want to delete it? [y|N] y  
Please wait while we import the service configuration...  
Service Configuration was imported.
```

In a site configuration, you perform this step only once.

If the password for **amadmin** has been changed through the OpenAM console, then use the bind password for the root DN of the configuration store.

6. Restart OpenAM.

In a site configuration, perform this step on all servers.

## Chapter 16

# Managing Certificates

This chapter shows you how to handle certificates used to protect network communication and for authentication.

In theory, you should not have to concern yourself with certificates when working with OpenAM. OpenAM core services and J2EE policy agents depend on the certificates installed for use with the web application container in which they run. OpenAM web policy agents depend on the certificates installed for use with the web server. Theoretically, each certificate has been signed by a well-known Certificate Authority (CA), whose certificate is already installed in the Java CA certificates trust store (`$JAVA_HOME/jre/lib/security/cacerts`, default password `changeit`) and in browsers, and so is recognized by other software used without you having to configure anything.

In practice, you might not have the budget for CA signed certificates in your lab or test environment, where you might constantly be installing new configurations, using and throwing away certificates for experiments and repeated tests. In the lab, therefore, you set up OpenAM to use self-signed certificates that you generate at no cost, but that are not recognized, and therefore not trusted out of the box.

How you configure the containers where OpenAM and your applications run to use self-signed certificates depends on your web application server or web server software. Yet, the basic principles apply.

- First, your container requires its own certificate for setting up secure connections.
- Second, the clients connecting must be able to trust the container certificate. Generally this means that clients must recognize the container certificate because they have a copy of the public certificate stored somewhere the client trusts.
- Third, if you use certificate authentication in OpenAM, OpenAM must also be able to find a copy of the client's public certificate to trust the client, most likely by finding a match with the certificate stored in the client profile from the identity repository. How you include client certificates in their identity repository entries depends on your identity repository more than it depends on OpenAM.

Some client applications let you trust certificates blindly. This can be helpful when working in your lab or test environment with self-signed certificates. For example, you might want to use HTTPS with the OpenAM RESTful API without having the client recognize the self-signed server certificate.

```
$ curl "https://openam.example.com:8443/openam/identity/authenticate?
username=bjensen&password=hifalutin"
curl: (60) Peer certificate cannot be authenticated with known CA certificates
More details here: http://curl.haxx.se/docs/sslcerts.html

curl performs SSL certificate verification by default, using a "bundle"
of Certificate Authority (CA) public keys (CA certs). If the default
bundle file isn't adequate, you can specify an alternate file
using the --cacert option.
If this HTTPS server uses a certificate signed by a CA represented in
the bundle, the certificate verification probably failed due to a
problem with the certificate (it might be expired, or the name might
not match the domain name in the URL).
If you'd like to turn off curl's verification of the certificate, use
the -k (or --insecure) option.
$ curl --insecure "https://openam.example.com:8443/openam/identity/authenticate?
username=bjensen&password=hifalutin"
token.id=AQIC5wM2LY4SfczMax8jegpSiaigB96N0WyllLilsd0PUMjY.*AAJTSQACMDE.*
```

### Procedure 16.1. To Set Up OpenAM With HTTPS on Tomcat

The container where you install OpenAM requires a certificate in order to set up secure connections. The following steps demonstrate one way to set up Tomcat with an HTTPS connector, using the Java **keytool** command to manage the certificate and key stores. Once Tomcat can do HTTPS, you deploy OpenAM as you normally would, over HTTPS.

1. Stop Tomcat.
2. Create a certificate and store it in a new key store.

```
$ cd /path/to/tomcat/conf/
$ keytool -genkey -alias openam.example.com -keyalg RSA -keystore keystore.jks
Enter keystore password:
What is your first and last name?
 [Unknown]: openam.example.com
What is the name of your organizational unit?
 [Unknown]: Eng
What is the name of your organization?
 [Unknown]: ForgeRock.com
What is the name of your City or Locality?
 [Unknown]: Grenoble
What is the name of your State or Province?
 [Unknown]: Isere
What is the two-letter country code for this unit?
 [Unknown]: FR
Is CN=openam.example.com, OU=Eng, O=ForgeRock.com, L=Grenoble, ST=Isere,
C=FR correct?
 [no]: yes

Enter key password for <openam.example.com>
(RETURN if same as keystore password):
```

3. Uncomment the SSL connector configuration in Tomcat's `conf/server.xml`, specifying your key store file and password.

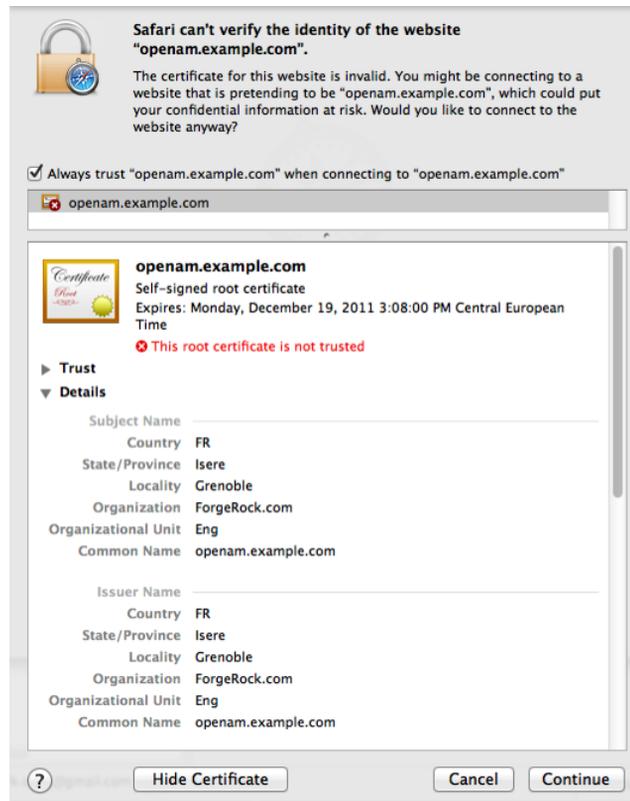
```

<!-- Define a SSL HTTP/1.1 Connector on port 8443
This connector uses the JSSE configuration, when using APR, the
connector should be using the OpenSSL style configuration
described in the APR documentation -->
<!--
-->
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
maxThreads="150" scheme="https" secure="true"
keystoreFile="/path/to/tomcat/conf/keystore.jks"
keystorePass="changeit"
clientAuth="false" sslProtocol="TLS" />

```

4. Start Tomcat.
5. Verify that you can connect to Tomcat on port 8443 over HTTPS.

Your browser does not trust the certificate, because the certificate is self-signed, not signed by any of the CAs your browser knows.



You recognize the Subject and Issuer of your certificate, and so can choose to trust the certificate, effectively saving it into your browser's trust store.

6. Deploy and configure OpenAM as you normally would.

### Procedure 16.2. To Share Self-Signed Certificates

When you use a self-signed certificate for your container, clients connecting must be able to trust the container certificate. Your browser makes this an easy, but manual process. For other client applications, you must import the certificate into the trust store used by the client. By default, Java applications can use the `$JAVA_HOME/jre/lib/security/cacerts` store. The default password is `changeit`.<sup>1</sup> The steps that follow demonstrate how to import a self-signed certificate into the Java `cacerts` store.

1. Export the certificate from the key store.

```
$ cd /path/to/tomcat/conf/
$ keytool -exportcert -alias openam.example.com -file openam.crt -keystore
  keystore.jks
Enter keystore password:
Certificate stored in file <openam.crt>
```

2. Import the certificate into the trust store.

```
$ keytool -importcert -alias openam.example.com -file openam.crt
  -trustcacerts -keystore $JAVA_HOME/jre/lib/security/cacerts
Enter keystore password:
Owner: CN=openam.example.com, OU=Eng, O=ForgeRock.com, L=Grenoble, ST=Isere,
  C=FR
Issuer: CN=openam.example.com, OU=Eng, O=ForgeRock.com, L=Grenoble, ST=Isere,
  C=FR
Serial number: 4e789e40
Valid from: Tue Sep 20 16:08:00 CEST 2011 until: Mon Dec 19 15:08:00 CET 2011
Certificate fingerprints:
  MD5: 31:08:11:3B:15:75:87:C2:12:08:E9:66:00:81:61:8D
  SHA1: AA:90:2F:42:0A:F4:A9:A5:0C:90:A9:FC:69:FD:64:65:D9:78:BA:1D
  Signature algorithm name: SHA1withRSA
  Version: 3
Trust this certificate? [no]: yes
Certificate was added to keystore
```

### Procedure 16.3. To Change the Signing Key for Federation

The following steps cover how to change the signing key for an identity provider. This procedure involves creating a self-signed certificate in a new key store file, and also preparing encrypted password files so that OpenAM can access the key store and the private key.

1. (Optional) If you do not already have the new signing key in your key store, generate a new key and key store.

The following example starts an interactive **keytool** session that requests information needed to generate a new key valid for two years, and puts it in a key store named `keystore.jks`. You

<sup>1</sup>Alternatively, you can specify the trust store for a Java application, such as `-Djavax.net.ssl.trustStore=/path/to/truststore.jks -Djavax.net.ssl.trustStorePassword=changeit`.

can perform this step in a temporary location, and then move the files generated once you have completed your work.

Keep track of the passwords you enter here, as you use them in the next step.

```
$ cd /tmp
$ keytool -genkeypair -alias newkey -keyalg RSA -keysize 1024 -validity 730
  -storetype JKS -keystore keystore.jks
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]: openam.example.com
What is the name of your organizational unit?
  [Unknown]: Eng
What is the name of your organization?
  [Unknown]: ForgeRock.com
What is the name of your City or Locality?
  [Unknown]: Grenoble
What is the name of your State or Province?
  [Unknown]: Isere
What is the two-letter country code for this unit?
  [Unknown]: FR
Is CN=openam.example.com, OU=Eng, O=ForgeRock.com, L=Grenoble, ST=Isere,
C=FR correct?
  [no]: yes

Enter key password for <newkey>
(RETURN if same as keystore password):
Re-enter new password:
```

Self-signed keys are not automatically recognized by other entities. You must also share the self-signed key as described in Procedure 16.2, "To Share Self-Signed Certificates".

- Using the passwords you entered in the previous step, prepare the password files to encrypt.

Create two files, each containing only a password in clear text. You can create the files in the same directory as the key store.

- `keypass.cleartext` contains the clear text key password for the private key you generated.
  - `storepass.cleartext` contains the clear text key store password.
- (Optional) If you have not already done so, install the administration tools as described in *To Set Up Administration Tools* in the *Installation Guide*.
  - Prepare encrypted password files for use by OpenAM.

```
$ ./ampassword --encrypt keypass.cleartext > .keypass
$ ./ampassword --encrypt storepass.cleartext > .storepass
```

Remove the `*.cleartext` files after preparing the encrypted versions.

- Replace the default OpenAM key store and password files with the ones that you have created.

The following example works with an installation of OpenAM where the deployment URI is `/openam`.

```
$ cp keystore.jks .keypass .storepass ~/openam/openam/
```

6. Restart OpenAM, or the container where it runs, so that OpenAM can use the new key store and encrypted password files.
7. Login to OpenAM console as administrator, and then set the new signing key in one of two ways.
  - a. If you have not yet configured your identity provider, select Common Tasks > Create Hosted Identity Provider, and then follow the instructions, selecting your key from the Signing Key drop-down list.
  - b. If you have already configured your identity provider, browse to Federation > *provider-name* > Assertion Content > Signing and Encryption, and then edit the signing key certificate alias.  
Save your work.
8. Share updated metadata with other entities in your circle of trust as described in *Setting Up SAML 2.0 SSO*.

## Chapter 17

# Monitoring OpenAM Services

This chapter covers how to monitor OpenAM services to ensure appropriate performance and service availability.

## 17.1. Monitoring Interfaces

OpenAM lets you monitor OpenAM over protocol through web pages, Java Management Extensions (JMX), or Simple Network Management Protocol (SNMP). The services are based on JMX.

To configure monitoring services, login to OpenAM console as OpenAM administrator, and browse to Configuration > System > Monitoring. Alternatively you can use the **ssoadm set-attr-defs** command.

```
$ ssoadm
set-attr-defs
--servicename iPlanetAMMonitoringService
--schematype Global
--adminid amadmin
--password-file /tmp/pwd.txt
--attributevalues iplanet-am-monitoring-enabled=true
```

Restart OpenAM for the changes to take effect. You must also restart OpenAM if you disable monitoring.

### 17.1.1. Web Based Monitoring

You can configure OpenAM to allow you to access a web based view of OpenAM MBeans on port 8082 where the core server runs, such as <http://openam-ter.example.com:8082/>. Either use the console, or use the **ssoadm** command.

```
$ ssoadm
set-attr-defs
--servicename iPlanetAMMonitoringService
--schematype Global
--adminid amadmin
--password-file /tmp/pwd.txt
--attributevalues iplanet-am-monitoring-http-enabled=true
```

The default authentication file allows you to authenticate over HTTP as user **demo**, password **changeit**. The user name and password are kept in the file specified, with the password encrypted.

```
$ cat openam/openam/openam_mon_auth
demo AQICMBCKLwx6G3vzK3TYRbtTpNYAagVIPNP
```

Or

```
$ cat openam/openam/opensso_mon_auth
demo AQICvSe+tXeg8TUUT8ekzHb8IRzVSvm1Lc2u
```

You can encrypt a new password using the **ampassword** command. After changing the authentication file, you must restart OpenAM for the changes to take effect.

## MBean View

[Project OpenDMKopendmk-1.0-b02]

- **MBean Name:**  
SUN\_OPENSSO\_SERVER\_MIB/ssoServerServerTable:ssoServerServerEntry.serverHostName=openam-ter.example.com,ssoServerServerEntry.serverPort=8080
- **MBean Java Class:** com.sun.identity.monitoring.SsoServerServerEntryImpl

Reload Period in seconds:

[Back to Agent View](#)

### MBean description:

Information on the management interface of the MBean

### List of MBean attributes:

Name	Type	Access	Value
<a href="#">ServerHostName</a>	java.lang.String	RO	openam-ter.example.com
<a href="#">ServerId</a>	java.lang.Integer	RO	1
<a href="#">ServerPort</a>	java.lang.Integer	RO	8080
<a href="#">ServerProtocol</a>	java.lang.String	RO	http
<a href="#">ServerStatus</a>	java.lang.Integer	RO	1

## 17.1.2. JMX Monitoring

You can configure OpenAM to allow you to listen for Java Management eXtension (JMX) clients, by default on port 9999. Either use the OpenAM console page under Configuration > System > Monitoring and make sure both Monitoring Status and Monitoring RMI interface status are both set to Enabled, or use the **ssoadm** command.

```
$ ssoadm
set-attr-defs
--servicename iPlanetAMMonitoringService
--schematype Global
--adminid amadmin
--password-file /tmp/pwd.txt
--attributevalues iplanet-am-monitoring-enabled=true
iplanet-am-monitoring-rmi-enabled=true
```

A number of tools support JMX, including **jvisualvm** and **jconsole**. When you use **jconsole** to browse OpenAM MBeans for example, the default URL for the OpenAM running on the local system is `service:jmx:rmi:///jndi/rmi://localhost:9999/server`.

```
$ jconsole service:jmx:rmi:///jndi/rmi://localhost:9999/server &
```

You can also browse the MBeans by connecting to your web application container, and browsing to the OpenAM MBeans. By default, JMX monitoring for your container is likely to be accessible only locally, using the process ID.

The screenshot shows the Java Monitoring & Management Console (jconsole) interface. The title bar reads "Java Monitoring & Management Console". The main window title is "pid: 14819 org.apache.catalina.startup.Bootstrap start". The interface is divided into two main sections: a left-hand tree view and a right-hand details pane.

The left-hand tree view shows a hierarchy of MBeans. The selected MBean is "openam-ter.exa" with a port number of "8080". Underneath it, there are several attributes listed: "ServerPort", "ServerId", "ServerStat", "ServerHost", and "ServerProt".

The right-hand pane displays the "MBeanInfo" for the selected MBean. It is organized into several sections:

- Info:** A table with two columns: "Name" and "Value".
 

ObjectName	SUN_OPENSSO_SERVER_MIB/ssoServerServerTable:ssoServerServerEntry.serverHostName=0...
ClassName	com.sun.identity.monitoring.SsoServerServerEntryImpl
Description	Information on the management interface of the MBean
- Constructor-0:** A table with two columns: "Name" and "Description".
 

Name	com.sun.identity.monitoring.SsoServerServerEntryImpl
Description	Public constructor of the MBean
- Parameter-0-0:** A table with two columns: "Name" and "Description".
 

Name	p1
Description	
- Type:** A single row: "Type" with value "com.sun.management.snmp.agent.SnmpMib".
- Descriptor:** A table with two columns: "Name" and "Value".
 

immutableInfo	true
interfaceClassName	com.sun.identity.monitoring.SsoServerServerEntryMBean
mxbean	false

Also see [Monitoring and Management Using JMX](#) for instructions on how to connect remotely, how to use SSL, and so forth.

### 17.1.3. SNMP Monitoring

You can configure OpenAM to allow you to listen on port 8085 for SNMP monitoring. Either use the console, or use the **ssoadm** command.

```
$ ssoadm
  set-attr-defs
  --servicename iPlanetAMMonitoringService
  --schematype Global
  --adminid amadmin
  --password-file /tmp/pwd.txt
  --attributevalues iplanet-am-monitoring-snmp-enabled=true
```

## 17.2. Is OpenAM Running?

You can check over HTTP whether OpenAM is up, using `isAlive.jsp`. Point your application to the file under the OpenAM URL, such as `http://openam.example.com:8080/openam/isAlive.jsp`.

If you get a success code (with `Server is ALIVE:` in the body of the page returned), then OpenAM is in operation.

## 17.3. Log Management

OpenAM implements logging as a service. This means remote clients such as your OpenAM policy agents can log messages to the central logging service.

### 17.3.1. Logging in OpenAM Core Services

By default OpenAM logs to files in the configuration directory for the instance, such as `$HOME/openam/log/` for log files, and `$HOME/openam/debug/` for debug files. You can also configure OpenAM to log through JDBC to a database such as MySQL or Oracle DB.

OpenAM sends messages to different log files, each named after the service logging the message, with two different types log files per service: `.access` and `.error`. Thus the current log files for the authentication service are named `amAuthentication.access` and `amAuthentication.error`.

See the [Log Messages](#) in the *Reference* reference for details.

OpenAM lets you change the log level on the fly. OpenAM also supports log rotation, secure logging, and log message buffering.

To configure OpenAM logging properties overall, login to the OpenAM console as OpenAM administrator, and browse to Configuration > System > Logging.

To adjust the debug level while OpenAM is running, login to the OpenAM console as OpenAM administrator, and browse to Configuration > Servers and Sites > *Server Name* > General, and then scroll down to Debugging. The default level for debug logging is Error. This level is appropriate for normal production operations, in which case no debug log messages are expected.

Setting debug log level to Warning increases the volume of messages. Setting debug log level to Message dumps detailed trace messages. Unless told to do so by qualified support personnel, do not use Warning and Message levels in production.

During development, you might find it useful to log all debug messages to a single file. In order to do so, set Merge Debug Files to `on`.

After changing this setting, restart OpenAM or the container in which it runs for the change to take effect.

### 17.3.2. Logging in OpenAM Policy Agents

By default, OpenAM Policy Agents log to local files in their configuration directories for debugging. The exact location depends on where you installed the agent.

By default OpenAM policy agents send log messages remotely to OpenAM when you log auditing information about URL access attempts. To configure audit logging for a centrally managed policy agent, login to the OpenAM console as administrator, and browse to Access Control > *Realm Name* > Agents > *Agent Type* > *Agent Name* > Global, and then scroll down to the Audit section.

### 17.3.3. Debug Logging by Service

OpenAM lets you capture debug log messages selectively for a specific service. This can be useful when you must turn on debugging in a production system where you want to avoid excessive logging, but must gather messages when you reproduce a problem.

Perform these steps to capture debug messages for a specific service.

1. Login to OpenAM console as administrator, `amadmin`.
2. Browse to `Debug.jsp`, for example `http://openam.example.com:8080/openam/Debug.jsp`.

No links to this page are provided in the console.

3. Select the service to debug and also the level required given the hints provided in the `Debug.jsp` page.

The change takes effect immediately.

4. Promptly reproduce the problem you are investigating.
5. After reproducing the problem, immediately return to the `Debug.jsp` page, and revert to normal log levels to avoid filling up the disk where debug logs are stored.

### 17.3.4. Rotating Debug Logs

By default OpenAM does not rotate debug logs. To rotate debug logs, edit `WEB-INF/classes/debugconfig.properties` where OpenAM is deployed.

The `debugconfig.properties` file includes the following properties.

#### `org.forgerock.openam.debug.prefix`

This property specifies the debug log file prefix applied when OpenAM rotates a debug log file. The property has no default. It takes a string as the property value.

#### `org.forgerock.openam.debug.suffix`

This property specifies the debug log file suffix applied when OpenAM rotates a debug log file. The property takes a `SimpleDateFormat` string. The default is `-MM.dd.yyyy-kk.mm`.

#### `org.forgerock.openam.debug.rotation`

This property specifies an interval in minutes between debug log rotations. Set this to a value greater than zero to enable debug log rotation.

After you edit the `debugconfig.properties` file, you must restart OpenAM or the web container where it runs for the changes to take effect.

## 17.4. Session Management

OpenAM console lets the administrator view and manage current user sessions under the Sessions tab page.

The screenshot shows the OpenAM console interface. At the top, there are navigation tabs: Common Tasks, Access Control, Federation, Configuration, and Sessions. The Sessions tab is selected. Below the tabs, there is a 'Current Sessions' section with a search bar and a 'Search' button. To the right, there is a 'View:' dropdown menu showing 'openam.example.com:8080'. Below this, there is a 'Sessions (2 Item(s))' section with an 'Invalidate Session' button. A table displays the session details for two users: amadmin and demo. The table has columns for User Id, Time Remaining (minutes), Max Session Time (minutes), Time Idle (minutes), and Max Idle Time (minutes). Each row has a checkbox in the first column.

<input checked="" type="checkbox"/>	User Id	Time Remaining (minutes)	Max Session Time (minutes)	Time Idle (minutes)	Max Idle Time (minutes)
<input type="checkbox"/>	amadmin	119	120	0	30
<input type="checkbox"/>	demo	119	120	0	30

To end a user session manually, select the user's session, and then click the Invalidate Session button. As a result, the user has to authenticate again.

## Chapter 18

# Tuning OpenAM

This chapter covers key OpenAM tunings to ensure smoothly performing access and federation management services, and to maximize throughput while minimizing response times.

### Note

The recommendations provided here are guidelines for your testing rather than hard and fast rules for every situation. Said another way, the fact that a given setting is configurable implies that no one setting is right in all circumstances.

The extent to which performance tuning advice applies depends to a large extent on your requirements, on your workload, and on what resources you have available. Test suggestions before rolling them out into production.

As a rule of thumb, an OpenAM server in production with a 3 GB heap can handle 100,000 sessions. Although you might be tempted to use a larger heap with a 64-bit JVM, smaller heaps are easier to manage. Thus, rather than scaling single servers up to increase the total number of simultaneous sessions, consider scaling out by adding more servers instead. The suggestions that follow pertain to production servers.

## 18.1. OpenAM Server Settings

OpenAM has a number of settings that can be tuned to increase performance.

### 18.1.1. General Settings

The following general points apply.

- Set debug level to `error`.
- Disable session failover debugging.
- Set container-level logging to a low level such as `error` or `severe`.

### 18.1.2. LDAP Settings

Tune both your LDAP data stores and also your LDAP authentication modules.

To change LDAP data store settings, browse to Access Control > *Realm Name* > Data Stores > *Data Store Name* in the OpenAM console. Each data store has its own connection pool and therefore each data store needs its own tuning.

*Table 18.1. LDAP Data Store Settings*

Property	Default Value	Suggestions
LDAP Connection Pool Minimum Size	1	The minimum LDAP connection pool size; a good tuning value for this property is 10.  ( <code>sun-idrepo-ldapv3-config-connection_pool_min_size</code> )
LDAP Connection Pool Maximum Size	10	The maximum LDAP connection pool size; a high tuning value for this property is 65, though you might well be able to reduce this for your deployment. Ensure your LDAP server can cope with the maximum number of clients across all the OpenAM servers.  ( <code>sun-idrepo-ldapv3-config-connection_pool_max_size</code> )

To change connection pool settings for the LDAP authentication module, browse to Configuration > Authentication > Core in the OpenAM console.

*Table 18.2. LDAP Authentication Module Setting*

Property	Default Value	Suggestions
Default LDAP Connection Pool Size	1:10	The minimum and maximum LDAP connection pool used by the LDAP authentication module. This should be tuned to 10:65 for production.  ( <code>iplanet-am-auth-ldap-connection-pool-default-size</code> )

### 18.1.3. Notification Settings

OpenAM has two thread pools used to send notifications to clients. The Service Management Service thread pool can be tuned in OpenAM console under Configuration > Servers and Sites > Default Server Settings > SDK.

*Table 18.3. SMS Notification Setting*

Property	Default Value	Suggestions
Notification Pool Size	10	This is the size of the thread pool used to send notifications. In production this value should be fine unless lots of clients are registering for SMS notifications.  ( <code>com.sun.identity.sm.notification.threadpool.size</code> )

The session service has its own thread pool to send notifications. This is configured under Configuration > Servers and Sites > Default Server Settings > Session.

*Table 18.4. Session Service Notification Settings*

Property	Default Value	Suggestions
Notification Pool Size	10	This is the size of the thread pool used to send notifications. In production this should be around 25-30.  ( <code>com.ipplanet.am.notification.threadpool.size</code> )
Notification Thread Pool Threshold	5000	This is the maximum number of notifications in the queue waiting to be sent. The default value should be fine in the majority of installations.  ( <code>com.ipplanet.am.notification.threadpool.threshold</code> )

#### 18.1.4. Session Settings

The session service has additional properties to tune, which are configured under Configuration > Servers and Sites > Default Server Settings > Session.

*Table 18.5. Session Settings*

Property	Default Value	Suggestions
Maximum Sessions	5000	In production this value can safely be set into the 100,000s. The maximum session limit is really controlled by the maximum size of the JVM heap which must be tuned appropriately to match the expected number of concurrent sessions.  ( <code>com.ipplanet.am.session.maxSessions</code> )
Sessions Purge Delay	0	This should be zero to ensure sessions are purged immediately.  ( <code>com.ipplanet.am.session.purgedelay</code> )

## 18.2. Java Virtual Machine Settings

This section gives some initial guidance on configuring the JVM for running OpenAM. These settings provide a strong foundation to the JVM before a more detailed garbage collection tuning exercise, or as best practice configuration for production.

Table 18.6. Heap Size Settings

JVM Parameters	Suggested Value	Description
<code>-Xms</code> & <code>-Xmx</code>	At least 1024m (2048m with embedded OpenDJ), in production environments at least 2048m to 3072m. This setting depends on the available physical memory, and on whether a 32 or 64-bit JVM is used.	-
<code>-server</code>	-	Ensures the server JVM is used
<code>-XX:PermSize</code> & <code>-XX:MaxPermSize</code>	Set both to 256m	Controls the size of the permanent generation in the JVM
<code>-Dsun.net.client.defaultReadTimeout</code>	60000	Controls the read timeout in the Java HTTP client implementation  This applies only to the Sun/Oracle HotSpot JVM.
<code>-Dsun.net.client.defaultConnectTimeout</code>	High setting: 30000 (30 seconds)	Controls the connect timeout in the Java HTTP client implementation  When you have hundreds of incoming requests per second, reduce this value to avoid a huge connection queue.  This applies only to the Sun/Oracle HotSpot JVM.

Table 18.7. Garbage Collection Settings

JVM Parameters	Suggested Value	Description
<code>-verbose:gc</code>	-	Verbose garbage collection reporting
<code>-Xloggc:</code>	<code>\$(CATALINA_HOME)/logs/gc.log</code>	Location of the verbose garbage collection log file
<code>-XX:+PrintClassHistogram</code>	-	Prints a heap histogram when a SIGTERM signal is received by the JVM

JVM Parameters	Suggested Value	Description
<code>-XX:+PrintGCDetails</code>	-	Prints detailed information about garbage collection
<code>-XX:+PrintGCTimeStamps</code>	-	Prints detailed garbage collection timings
<code>-XX:+HeapDumpOnOutOfMemoryError</code>	-	Out of Memory errors generate a heap dump automatically
<code>-XX:HeapDumpPath</code>	<code>\$CATALINA_HOME/logs/heapdump.hprof</code>	Location of the heap dump
<code>-XX:+UseConcMarkSweepGC</code>	-	Use the concurrent mark sweep garbage collector
<code>-XX:+UseCMSCompactAtFullCollection</code>	-	Aggressive compaction at full collection
<code>-XX:+CMSClassUnloadingEnabled</code>	-	Allow class unloading during CMS sweeps

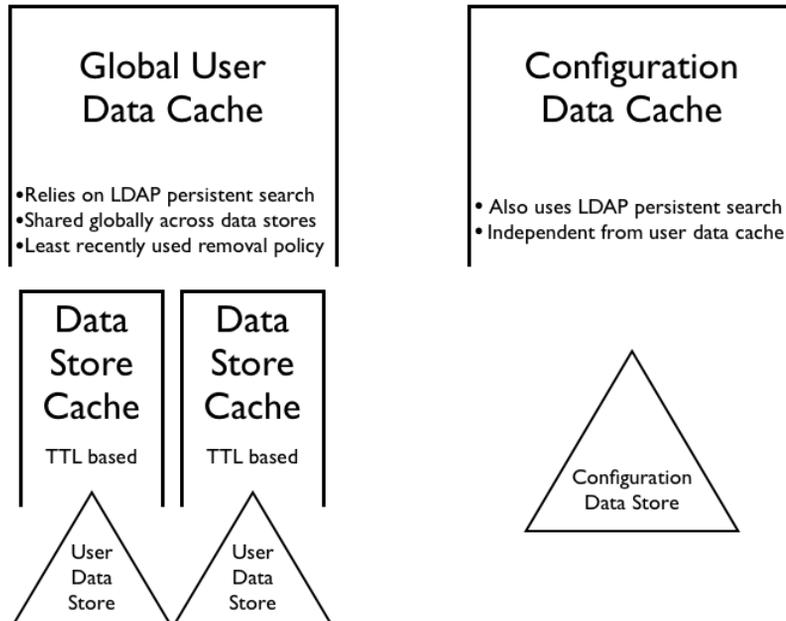
## 18.3. Caching in OpenAM

OpenAM caches data to avoid having to query user and configuration data stores each time it needs the information. By default, OpenAM makes use of LDAP persistent search to receive notification of changes to cached data. For this reason, caching works best when data are stored in a directory server that supports LDAP persistent search.

OpenAM has two kinds of cache on the server side that you can configure, one for configuration data and the other for user data. Generally use the default settings for configuration data cache. This section mainly covers the configuration choices you have for caching user data.

OpenAM server has two levels of user data caching. The global user data cache is dirtied by LDAP persistent search. When the global user data cache is enabled (as it is by default), the individual data store caches are not useful because all requests for data go through the global cache. The individual data store caches are therefore only useful when the global user data cache is disabled.

The following diagram depicts the two kinds of cache, and also the two levels of user data caching.



The rest of this section concerns mainly settings for global user data cache and for SDK clients. For a look at data store cache settings, see Table 18.1, "LDAP Data Store Settings".

### 18.3.1. Overall Server Cache Settings

By default OpenAM has caching enabled both for configuration data and also for user data. This setting is governed by the server property `com.iplanet.am.sdk.caching.enabled`, which by default is `true`. When you set this advanced property to `false`, then you can enable caching independently for configuration data and for user data.

#### *Procedure 18.1. To Turn Off Global User Data Caching*

**Disabling caching can have a severe negative impact on performance. This is because, when caching is disabled, OpenAM must query a data store each time it needs data.**

If, however, you have at least one user data store that does not support LDAP persistent search, such as a relational database or an LDAP directory server that does not support persistent search, then you must disable the *global* cache for user data. Otherwise user data caches cannot stay in sync with changes to user data entries.

1. In the OpenAM console, browse to Configuration > Servers and Sites > *Server Name* > Advanced.
2. Set `com.ipplanet.am.sdk.caching.enabled` to `false` to disable caching overall.
3. Set `com.sun.identity.sm.cache.enabled` to `true` to enable configuration data caching.

All supported configuration data stores support LDAP persistent search, so it is safe to enable configuration data caching.

You must explicitly set this property to `true`, because setting `com.ipplanet.am.sdk.caching.enabled` to `false` in the previous step disables both user and configuration data caching.

4. Save your work.
5. OpenAM starts persistent searches on user data stores when possible<sup>1</sup> in order to monitor changes. With user data store caching disabled, OpenAM still starts the persistent searches, even though it no longer uses the results.

Therefore, if you disable user data store caching, you should also disable persistent searches on user data stores in your deployment to improve performance. To disable persistent search on a user data store, remove the value of the Persistent Search Base DN configuration property and leave it blank. Locate this property under Realms > *Realm Name* > Data Stores > *Data Store Name* > Persistent Search Controls.

### Procedure 18.2. To Change the Maximum Size of Global User Data Cache

With a large user data store and active user base, the number of user entries in cache can grow large.

1. In the OpenAM console, browse to Configuration > Servers and Sites > Default Server Settings > SDK.
2. Change the value of SDK Caching Max. Size, and then Save your work.

There is no corresponding setting for configuration data, as the number of configuration entries in a large deployment is not likely to grow nearly as large as the number of user entries.

## 18.3.2. Caching Properties For Java EE Policy Agents & SDK Clients

Policy agents and other OpenAM SDK clients can also cache user data, using most of the same properties as OpenAM server as described in Table 18.8, "OpenAM Cache Properties". Clients however can receive updates by notification from OpenAM or, if notification fails, by polling OpenAM for changes.

---

<sup>1</sup> OpenAM starts persistent searches on user data stores on directory servers that support the `psearch` control.

### Procedure 18.3. To Enable Notification & Polling For Client Cache Updates

This procedure describes how to enable change notification and polling for policy agent user data cache updates. When configuring a custom OpenAM SDK client using a `.properties` file, use the same properties as for the policy agent configuration.

1. In OpenAM console, browse to Access Control > *Realm Name* > Agents > *Agent Type* > *Agent Name* to view and edit the policy agent profile.
2. On the Global tab page, check that the Agent Notification URL is set.

When notification is enabled, the agent registers a notification listener with OpenAM for this URL.

The corresponding property is `com.sun.identity.client.notification.url`.

3. For any changes you make, Save your work.

You must restart the policy agent for the changes to take effect.

### 18.3.3. Cache Settings

The table below provides a quick reference, primarily for user data cache settings.

Notice that many properties for configuration data cache have `sm` (for Service Management) in their names, whereas those for user data have `idm` (for Identity Management) in their names.

Table 18.8. OpenAM Cache Properties

Property	Description	Default	Applies To
<code>com.ipplanet.am.sdk.cache.maxSize</code>	Maximum number of user entries cached	10000	Server & SDK
<code>com.ipplanet.am.sdk.caching.enabled</code>	Whether to enable caching for both configuration data and also for user data.  If <code>true</code> , this setting overrides <code>com.sun.identity.idm.cache.enabled</code> and <code>com.sun.identity.sm.cache.enabled</code> .  If <code>false</code> , you can enable caching independently for configuration data and for user data using the aforementioned properties.	<code>true</code>	Server & SDK
<code>com.ipplanet.am.sdk.remote.pollingTime</code>	How often in minutes the SDK client such as a policy agent should poll OpenAM for modified user data entries.	1 (minute)	SDK

Property	Description	Default	Applies To
	<p>The SDK also uses this value to determine the age of the oldest changes requested. The oldest changes requested are 2 minutes older than this setting. In other words, by default the SDK polls for entries changed in the last 3 minutes.</p> <p>Set this to 0 or a negative integer to disable polling.</p>		
<code>com.sun.am.event.notification.expire.time</code>	How long OpenAM stores a given change to a cached entry, so that clients polling for changes do not miss the change.	30 (minutes)	Server only
<code>com.sun.identity.idm.cache.enabled</code>	<p>If <code>com.iplanet.am.sdk.caching.enabled</code> is <code>true</code>, this property is ignored.</p> <p>Otherwise, set this to <code>true</code> to enable caching of user data.</p>	<code>false</code>	Server & SDK
<code>com.sun.identity.idm.cache.entry.default.expire.time</code>	How many minutes to store a user data entry in the global user data cache	30 (minutes)	Server & SDK
<code>com.sun.identity.idm.cache.entry.expire.enabled</code>	Whether user data entries in the global user data cache should expire over time	<code>false</code>	Server & SDK
<code>com.sun.identity.idm.remote.notification.enabled</code>	<p>Whether the SDK client such as a policy agent should register a notification listener for user data changes with the OpenAM server.</p> <p>The SDK client uses the URL specified by <code>com.sun.identity.client.notification.url</code> to register the listener so that OpenAM knows where to send notifications.</p> <p>If notifications cannot be enabled for some reason, then the SDK client falls back to polling for changes.</p>	<code>true</code>	SDK
<code>com.sun.identity.sm.cache.enabled</code>	<p>If <code>com.iplanet.am.sdk.caching.enabled</code> is <code>true</code>, this property is ignored.</p> <p>Otherwise, set this to <code>true</code> to enable caching of configuration data. It is recommended that you always set this to <code>true</code>.</p>	<code>false</code>	Server & SDK

## Chapter 19

# Changing Host Names

When you change the OpenAM host name, you must make manual changes to the configuration. This chapter describes what to do. If you also must move an embedded configuration directory from one host to another, see the OpenDJ documentation on *Moving Servers*.

Changing OpenAM host names involves the following high level steps.

- Adding the new host name to the Realm/DNS Aliases list
- Exporting, editing, then importing the configuration

This step relies on the **ssoadm** command, which you install separately from OpenAM as described in *To Set Up Administration Tools* in the *Installation Guide*.

- Stopping OpenAM and editing configuration files
- Removing the old host name from the Realm/DNS Aliases list

Before you start, make sure you have a current backup of your current installation. See *Backing Up and Restoring OpenAM Configurations* for instructions.

### *Procedure 19.1. To Add the New Host Name As an Alias*

1. Login to OpenAM console as administrator, **amadmin**.
2. Under Access Control > / (Top Level Realm), add the new host name to the Realm/DNS Aliases list, and then save your work.

### *Procedure 19.2. To Export, Edit, & Import the Service Configuration*

1. Export the service configuration.

```
$ ssoadm export-svc-cfg -u amadmin -e fZatIu680iqccJMXosSRyVjMswJIx+SA  
-f /tmp/pwd.txt -o config.xml
```

Service Configuration was exported.

In this example, the secret key for encrypting the password in `-e fZatIu680iqccJMXosSRyVjMswJIx+SA` was taken from the Password Encryption Key field in the OpenAM console under Configuration > Servers and Sites > *Server Name* > Security.

2. Edit the service configuration file.

- Change the fully qualified domain name, such as `openam.example.com`, throughout the file.
- If you are changing the deployment descriptor, such as `/openam`, then change the value of `com.iplanet.am.services.deploymentDescriptor`.

Also change the deployment descriptor in the `propertiesViewBeanURL="deployment-descriptor/auth/ACServiceInstanceList"` attribute.

Also change the deployment descriptor in the `propertiesViewBeanURL="deployment-descriptor/auth/ACModuleList"` attribute.

Also change the deployment descriptor in a `<Value>` element that is a child of an `<AttributeValuePair>` element.

Also change the deployment descriptor where it occurs throughout the file in the full URL to OpenAM, such as `http://openam.example.com:8080/deployment-descriptor`.

- If you are changing the port number, then change the value of `com.iplanet.am.server.port`.

Also change the port number in `host:port` combinations throughout the file.

- If you are changing the domain name, then change the cookie domain such as `<Value>.example.com</Value>` throughout the file.

### 3. Import the updated service configuration.

```
$ ssoadm import-svc-cfg -u amadmin -e fZatIu680iqccJMXosSRyVjMswJIx+SA  
-f /tmp/pwd.txt -X config.xml
```

```
Directory Service contains existing data. Do you want to delete it? [y|N] y  
Please wait while we import the service configuration...  
Service Configuration was imported.
```

## Procedure 19.3. To Edit OpenAM Configuration Files For the New Host Name

1. Stop the web container where OpenAM runs.
2. Edit the bootstrap file, such as `/home/user/openam/bootstrap`, changing the FQDN, port, and deployment descriptor for OpenAM as necessary.
3. If you are changing the deployment descriptor, then move the folder containing OpenAM configuration, such as `/home/user/openam/`, to match the new deployment descriptor, such as `/home/user/openam2/`.
4. If you are changing the location or deployment descriptor, change the name of the file in the `/home/user/.openamcfg` folder, such as `AMConfig_path_to_tomcat_webapps_openam` to match the new location and deployment descriptor.

Also edit the path name in the file to match the change you made when moving the folder.

5. Restart the web container where OpenAM runs.

*Procedure 19.4. To Remove the Old Host Name As an Alias*

1. Login to OpenAM console as administrator, `amadmin`.
2. Under Access Control > / (Top Level Realm), remove the old host name from the Realm/DNS Aliases list, and then save your work.

## Chapter 20

# Securing OpenAM

This chapter identifies best practices for securing your OpenAM deployment.

## 20.1. Avoiding Obvious Defaults

OpenAM includes default settings to make it easier for you to evaluate the software. Avoid these default settings in production deployments.

- When connecting to LDAP, bind with a specific administrative account rather than a root DN account if possible.
- Change the default `iPlanetDirectoryPro` cookie name both in OpenAM (`com.iplanet.am.cookie.name`) and in your policy agent profiles (`com.sun.identity.agents.config.cookie.name`).
- When installing OpenAM, do not use `/openam` or `/opensso` as the deployment URI.
- Set valid goto URL domains for OpenAM in the core authentication module configuration. The parameter is described in the section providing *Hints For the Core Authentication Module* (`iplanet-am-auth-valid-goto-domains`).
- Create an administrator in the top-level realm with a different ID than the default `amadmin`.
- Create specific administrator users to track better who makes configuration changes.
- Set the OpenAM advanced property `openam.auth.soap.rest.generic.authentication.exception` to `true`. This causes OpenAM to return the same exception both when the user does not exist, and also when the password is not valid.

## 20.2. Protecting Network Access

Anytime users interact with a web service, there are risks. With OpenAM, you can reduce those risks by deploying different parts of OpenAM in appropriate parts of an enterprise network.

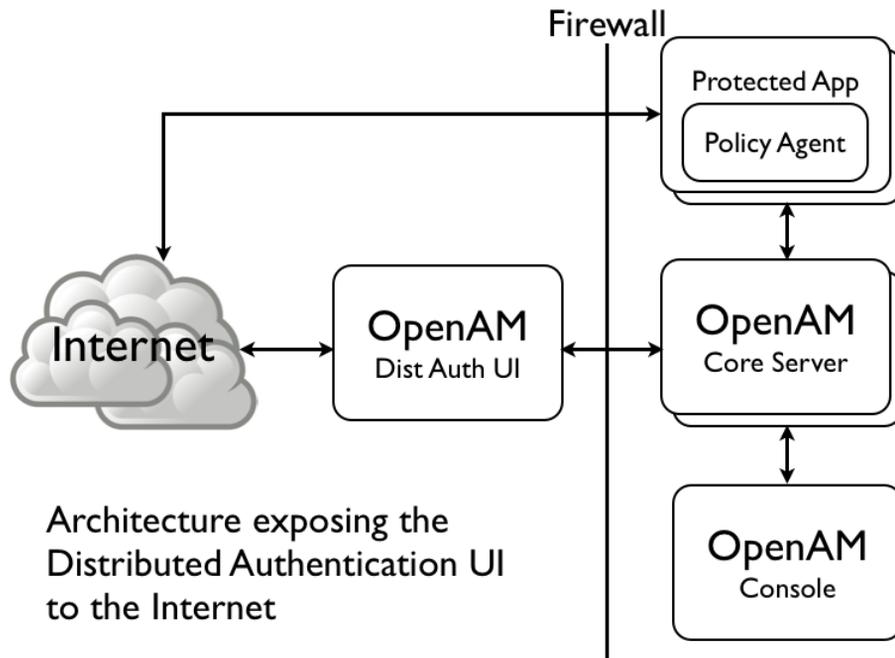
To minimize risks, deploy only the core OpenAM server on systems directly connected through a firewall. As a start, deploy only the core server (and the protected web application) on Internet-facing servers. For instructions, see the following section from the OpenAM Installation Guide, *Determine Which War File to Deploy* in the *Installation Guide*.

You can further limit what is exposed through the firewall by using one of two strategies:

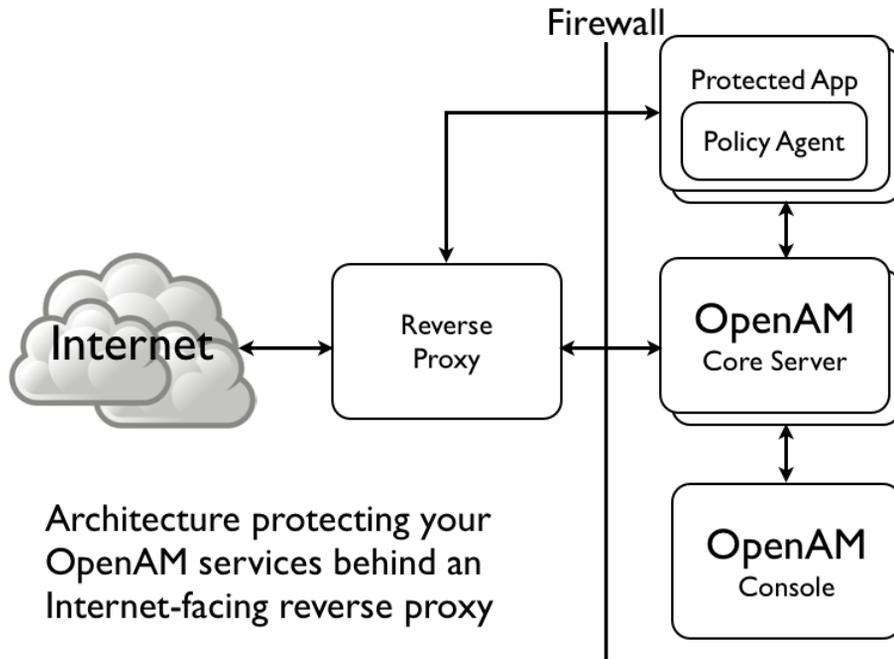
- Set up a distributed authentication user interface (UI) in a DMZ between firewalls.

The distributed authentication UI is essentially a small subset of the OpenAM server with just enough login logic to receive user authentication requests. Those requests are forwarded to the core OpenAM servers.

See *Installing OpenAM Distributed Authentication* in the *Installation Guide* for installation instructions. The following figure shows the recommended architecture.



- Alternatively, use a reverse proxy in front of OpenAM to allow access only to the necessary URLs. The following figure shows the recommended architecture with a reverse proxy.



For access to the console, deploy the full OpenAM application<sup>1</sup> on a separate system that is reachable only from internal systems. Do not include the full OpenAM server in the load-balanced pool of OpenAM servers serving applications.

- Leave `ssoadm.jsp` disabled in production. (Advanced property: `ssoadm.disabled=true`)
- If possible in your deployment, control access to OpenAM console by network address, such that administrators can only connect from well-known systems and networks.
- Restrict access to URIs that you do not use, and prevent internal endpoints such as `/sessionservice` from being reachable over the Internet.

For a full list of endpoints, see the OpenAM Reference Guide Chapter on *Service Endpoints* in the *Reference*.

## 20.3. Securing OpenAM Administration

Keep administration of access management services separate from management of the services themselves.

<sup>1</sup>Console only deployment is no longer supported.

- Create realms for your organization(s) and separate administrative users from end users. For instructions, see *Configuring Realms*. You must then either:
  - Use the `realm=realm-name` query string parameter when redirecting users to OpenAM, which gives you a way to isolate the URLs used by an application.
  - Create fully qualified domain name realm/DNS aliases, and use them to control access to the realms.
- When customizing `config/auth/default*/Login.jsp`, make sure that you do not introduce any security vulnerabilities such as cross-site scripting due to unvalidated input.
- Create a policy agent profile for each policy agent. See *Configuring Policy Agent Profiles* for instructions.

## 20.4. Securing Communications

Keep communications secure by using encryption, properly configured cookies, and request and response signatures.

- Protect network traffic by using HTTPS and LDAPS where possible.
- When using HTTPS, use secure cookies.
- Where possible, use subdomain cookies, and control subdomains in a specific DNS master.
- Use cookie hijacking protection with restricted tokens, where each policy agent uses different SSO tokens for the same user. See *To Protect Against CDSSO Cookie Hijacking* for instructions.
- When using SAML 2.0:
  - Sign authentication requests, authentication responses, and single logout requests.
  - If the other entities in your circle of trust can handle encryption, then use encryption as well.
  - Use your own key, not the `test` key provided with OpenAM.

## 20.5. Administering the amadmin Account

You can make changes to the password and user name for the main OpenAM administrative account.

You can change the user name of the `amadmin` administrative account to something more obscure, such as `superroot`. However, the capabilities of that alternative administrative account would not be complete, due to some hard-coding of `amadmin` in the source files. When changing the password for the main OpenAM administrative account, you must make a corresponding change to the authentication datastore. That datastore could be OpenDJ. The steps you would take to change the OpenAM top-level administrative password and account name are shown in the following sections.

*Procedure 20.1. To Change the Password for the Top-Level Administrator (normally `amadmin`)*

1. Login to the OpenAM console as the administrator, normally `amadmin`.
2. Under Access Control > / (Top Level Realm) > Subjects > User, select the name of the current top-level administrative user.
3. In the page that appears, navigate to the Password row and click Edit.
4. In the window that appears, enter the desired new password in the New Password and Re-Enter Password text boxes.
5. Click OK to implement the change. If you want to cancel, click Close or just close the window.
6. You'll also need to change the password for the administrator on the directory server. If you are using OpenDJ, refer to the *OpenDJ Administration Guide* section on [Resetting Administrator Passwords](#). If you are using a different directory server, you will have to refer to the documentation for that server.

In the following steps, you will identify the new administrative user by assigning it to the `com.sun.identity.authentication.super.user` directive. You may also need to create an OpenAM account for the new administrative user. Don't forget to make sure that new administrative account is configured in the corresponding directory server such as OpenDJ.

*Procedure 20.2. To Change the Account Name for the Top-Level Administrator (normally `amadmin`)*

1. Login to the OpenAM console as the administrator, normally `amadmin`.
2. Navigate to the page where you can set the properties for different classes. Select Configuration > Servers and Sites > *Server Name* > Advanced.
3. In the Advanced Properties window that appears, click Add.
4. You'll see blank entries in the end of the list of Property Names and Property Values. In the empty Property Name text box, enter `com.sun.identity.authentication.super.user`.
5. In the corresponding Property Values text box, enter appropriate values for the new administrative user in LDAP Data Interchange Format (LDIF). For example, the following entry would set up an administrative user named `superroot`, in the organizational unit named `peoplepeople`, associated with the example.com domain: `uid=superroot,ou=people,dc=example,dc=com`.
6. Click Save to save the changes that you've made.
7. If the account doesn't already exist in OpenAM or on a connected directory server, you'll need to create it. To do so, select Access Control > / (Top Level Realm) > Subject > User > New. In the New User window that appears, create the new user. Make sure to enter an appropriate password and make that user Active. The ID for that new user is the user name.

8. As noted earlier, you'll also need to make sure that the corresponding account on the directory server has at least CN=Directory Manager privileges. If you're using OpenDJ, refer to the chapter on *Configuring Privileges & Access Control* in the *OpenDJ Administration Guide*.

If you do change the account name of the top-level administrative account, you should be aware that the original `amadmin` account is "hard-coded" in the source code of several files. The code in these files may affect the functionality of a top-level administrative user with a name other than `amadmin`.

One of the improvements that we plan to make to OpenAM is to eliminate these instances of hard-coding. Until we make such improvements, the `amadmin` user would retain privileges related to the `LoginState` and some IDM-related classes.

## Chapter 21

# Troubleshooting

This chapter covers how to get debugging information and troubleshoot issues in OpenAM deployments.

## Solutions to Common Issues

This section offers solutions to common problems when working with OpenAM.

### 21.1. OpenAM Installation

- Q:** OpenAM configuration could not write to the configuration directory. Where must I change permissions, and what permissions are required?
- A:** If the user running the web container has a \$HOME directory, then the configuration directory is stored there, and you probably do not have this problem. If you do not know the user running the web container, use the `ps` command to check. In the following example, the user is `mark`, the web container `tomcat`.

```
$ ps -ef | grep tomcat
mark      1739      1  0 14:47...
```

For a container installed from native packages with a dedicated user, \$HOME may not be where you think it is. Look at the user's entry in `/etc/passwd` to locate the home directory. The user running the web container where you install OpenAM must be able to read from and write in this directory.

If you cannot change the permissions to the user's home directory, you can, as a workaround, unpack `OpenAM-11.0.0.war`, set the `configuration.dir` property in the `WEB-INF/classes/bootstrap.properties` to a directory with appropriate permissions, and repack `openam.war` with the adjusted file before deploying that.

```
$ cd ~/Downloads/openam/OpenAM-11.0.0.war
$ mkdir unpacked ; cd unpacked
$ jar xf ../OpenAM-11.0.0.war
$ vi WEB-INF/classes/bootstrap.properties
$ grep ^config WEB-INF/classes/bootstrap.properties
configuration.dir=/my/readwrite/config/dir
$ jar cf ../openam.war *
```

- Q:** Deployment failed due to lack of memory. What do I do?

- A:** OpenAM requires at least a maximum heap size of 1024 MB, with a 256 MB maximum permanent generation heap size. For the Sun JVM, ensure the container starts with `-Xmx1024m -XX:MaxPermSize=256m` for these settings.

If you do not know the settings used when the web container was started, use the `ps` command to check. In the following example, the web container is `tomcat`.

```
$ ps -ef | grep tomcat | grep Xm
... -Xmx1024m -XX:MaxPermSize=256m ...
```

Make sure you have at least 2 GB of RAM on the system where you run OpenAM to avoid running out of memory.

If you make it through deployment and seem to be running out of memory later, you can confirm memory errors in OpenAM by searching the `config-dir/openam/debug/*` files for `java.lang.OutOfMemoryError`.

- Q:** Deployment failed due to invalid hostname configuration. What do I do?

- A:** OpenAM requires that you use a fully qualified domain name (FQDN) that the host can resolve.

```
$ ping openam-ter.example.com
PING openam-ter (192.168.56.2) 56(84) bytes of data.
64 bytes from openam (192.168.56.2): icmp_seq=1 ttl=64 time=0.025 ms
64 bytes from openam (192.168.56.2): icmp_seq=2 ttl=64 time=0.032 ms
64 bytes from openam (192.168.56.2): icmp_seq=3 ttl=64 time=0.030 ms
```

For a test deployment (at home, on a laptop), you can use fake FQDNs in `/etc/hosts` (`%SystemRoot%\system32\drivers\etc\hosts` on Windows), depending on how your network is configured.

```
$ cat /etc/hosts | grep openam
192.168.56.2 openam openam.example.com
192.168.56.3 openam-bis openam-bis.example.com
192.168.56.5 openam-ter openam-ter.example.com
```

- Q:** I configured OpenAM, and now am seeing the configuration screen again. Who deleted my configuration?
- A:** OpenAM uses a file in `$HOME/.openamcfg/` to bootstrap and find its configuration. The file is named after the path to OpenAM and contains the path to the configuration. The following example shows what the file looks like for OpenAM deployed in Apache Tomcat under `/path/to/tomcat/webapps/openam`, and running as user `amuser` with `$HOME /home/amuser`.

```
$ cat .openamcfg/AMConfig_path_to_tomcat_webapps_openam_
/home/amuser/openam
```

If OpenAM cannot find its configuration, then it displays the configuration screen.

## 21.2. OpenAM Upgrades

- Q:** I have upgraded OpenAM, now my tools are not working properly. What happened?

- A:** Every OpenAM component must be upgraded, not just the main OpenAM `.war` file. If you did not upgrade the tools too, they may not work as intended.

## 21.3. OpenAM Administration

- Q:** I cannot use the browser-based equivalent of `ssoadm`, `http://openam.example.com:8080/openam/ssoadm.jsp`. Why not?

- A:** For security reasons, `ssoadm.jsp` is not activated by default. To activate it, browse to Configuration > Servers and Sites > Servers > *ServerName* > Advanced, and then add a property named `ssoadm.disabled` with value `false`.

- Q:** The `ssoadm` command is very, very slow on my virtual machine (VMWare, VirtualBox, etc.). How can I speed it up?

- A:** Virtual machine random devices do not always produce enough random data. The `ssoadm` command can hang while reading random data from the virtual machine's random device, with the result that you can wait a minute or more for a single command to finish.

To work around this limitation on virtual machines, make sure you install something that generates enough random data, such as a timer entropy daemon.

- Q:** I added OpenDJ as a data store, and now I cannot add a user. OpenAM gives me the following error.

```
ERROR: LDAPv3Repo.create failed. errorCode=65 Entry
uid=test,ou=people,dc=example,dc=com violates the Directory Server
schema configuration because it includes attribute inetUserStatus which
is not allowed by any of the objectclasses defined in that
entry
```

- A:** When you set up a New Data Store to use OpenDJ as an identity repository under Access Control > *Realm Name* > Data Stores > New..., you need to check the Load schema when saved box if you want OpenAM to add the schema to OpenDJ. The box is not selected by default.

The full version of OpenAM includes directory server schema in the `~/Downloads/openam/ldif/` directory. To add the schema to OpenDJ afterwards, you can try the following command.

```
$ /path/to/OpenDJ/bin/ldapmodify
--port 1389
--bindDN "cn=Directory Manager"
--bindPassword password
--filename ~/Downloads/openam/ldif/fam_sds_schema.ldif
Processing MODIFY request for CN=schema
MODIFY operation successful for DN CN=schema
```

- Q:** I have session failover configured for an OpenAM site. I see many connections in `TIME_WAIT` state, and the connections seem to be used only for communication between OpenAM servers in that site. What should I set to have fewer connections in `TIME_WAIT`?

- A:** When you have session failover configured for a site, OpenAM servers run health checks against other servers in the same site. By default, the health checks are run every second (1000 milliseconds) with a timeout of 1 second (1000 milliseconds).

If there is network latency between servers in a site, for example if you are running your servers in virtual machines, the default settings might not be right for your deployment. In that case, consider changing the following advanced server properties.

- By lengthening `com.ipplanet.am.session.failover.cluster.stateCheck.timeout` and `com.ipplanet.am.session.failover.cluster.stateCheck.period` to something longer than the default, you can work around issues with network latency.
- By setting `com.sun.identity.urlchecker.dorequest` to `true` or `false`, you can change whether OpenAM performs an HTTP GET request or only checks the Socket connection of `com.sun.identity.urlchecker.targeturl` as a health check.

To set advanced properties, either use the OpenAM console page under Configuration > Servers and Sites > Default Server Settings > Advanced, or set the properties using the **ssoadm update-server-cfg** command as in the following example, which updates the default server configuration:

```
$ ./ssoadm update-server-cfg -s default -u amadmin -f /tmp/pwd.txt  
-a com.ipplanet.am.session.failover.cluster.stateCheck.timeout=2000
```

# OpenAM Glossary

Agent administrator	User having privileges only to read and write policy agent profile configuration information, typically created to delegate policy agent profile creation to the user installing a policy agent
Authentication	The act of confirming the identity of a principal
Authentication module	OpenAM authentication unit that handles one way of obtaining and verifying credentials
Authentication chaining	A series of authentication modules configured together which a principal must negotiate as configured in order to authenticate successfully
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
Agent authenticator	Entity with read-only access to multiple agent profiles defined in the same realm; allows an agent to read web service profiles
Application	In the context of OpenAM entitlements, protected resources that share a common set of actions and related policies
Attribute based access control (ABAC)	Access control that is based on attributes of a user, such as how old a user is or whether she is a paying customer
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

---

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. OpenAM can play this role in the OAuth 2.0 authorization framework.
Auto-federation	Arrangement to federate a principal's identity automatically based on a common attribute value shared across the principal's profiles at different providers
Bulk federation	Batch job permanently federating user profiles between a service provider and an identity provider based on a list of matched user identifiers that exist on both providers
Circle of trust	Group of providers, including at least one identity provider, who have agreed to trust each other to participate in a SAML 2.0 provider federation
Client	In OAuth 2.0, requests protected web resources on behalf of the resource owner given the owner's authorization. OpenAM can play this role in the OAuth 2.0 authorization framework.
Conditions	Optional conditions under which a policy applies
Configuration data store	LDAP directory service holding OpenAM configuration data
Cross-domain single sign on (CDSSO)	OpenAM capability allowing single sign on across different DNS domains
Delegation	In the context of OpenAM entitlements, a means of granting specific users privileges to manage policies
Entitlement	XACML-based policy
Extended metadata	Federation configuration information specific to OpenAM
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 OpenAM on the

---

	service provider side; OpenAM lets you create both .NET and Java Fedlets.
Hot swappable	Refers to configuration properties for which changes can take effect without restarting the container where OpenAM runs
Identity	Set of data that uniquely describes a person or a thing such as a device or an application
Identity federation	Linking of a principal's identity across multiple providers
Identity provider (IdP)	Entity that produces assertions about a principal (such as how and when a principal authenticated, or that the principal's profile has a specified attribute value)
Identity repository	Data store holding user profiles and group information; different identity repositories can be defined for different realms.
Java EE policy agent	Java web application installed in a web container that acts as a policy agent, filtering requests to other applications in the container with policies based on application resource URLs
Metadata	Federation configuration information for a provider
Policy	Set of rules that define who is granted access to a protected resource when, how, and under what conditions
Policy administration point (PAP)	Entity that manages and stores policy definitions
Policy agent	Agent that intercepts requests for resources, directs principals to OpenAM for authentication, and enforces policy decisions from OpenAM
Policy decision point (PDP)	Entity that evaluates access rights and then issues authorization decisions
Policy enforcement point (PEP)	Entity that intercepts a request for a resource and then enforces policy decisions from a PDP
Policy information point	Entity that provides extra information such as user profile attributes that a PDP needs in order to make a decision
Principal	Entity that can be authenticated (such as a user, a device, or an application)
Provider federation	Agreement among providers to participate in a circle of trust
Realm	OpenAM unit for organizing configuration and identity information, making it possible to delegate administration for part of OpenAM

---

	configuration; realms can be used for example when different parts of an organization using OpenAM have different policies and different identity repositories
Referral	Means to delegate policy management and decision making for a realm
Resource	Something a principal can access over the network such as a web page
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 providers	Policy extensions that define additional information to return in an authorization decision beyond "allow" or "deny"
Role based access control (RBAC)	Access control that is based on whether a user has been granted a set of permissions (a role)
Rules	Definitions identifying how a policy matches resources to which access is granted or denied
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	In OpenAM a user session is the interval that starts with the user authenticating through OpenAM and ends when the user logs out, or when her session is terminated. OpenAM manages user sessions across one or more applications by issuing a session token used to identify the session and by tracking the session state in order to handle session events like logout and timeout, to permit session constraints, and to notify applications involved in SSO when a session ends.
Session failover	Capability to allow another OpenAM server to manage a session when the OpenAM server that initially authenticated the principal goes offline
Session token	Unique identifier issued by OpenAM after successful authentication, 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
Standard metadata	Standard federation configuration information that you can share with other access management software
Subject	Entity that can request access to a resource
User data store	Data storage service holding principals' profiles; underlying storage can be an LDAP directory service, a relational database, or a custom <b>IdRepo</b> implementation
Web policy agent	Native library installed in a web server that acts as a policy agent with policies based on web page URLs

# Appendix A. Release Levels & Interface Stability

This appendix includes ForgeRock definitions for product release levels and interface stability.

## A.1. ForgeRock Product Release Levels

ForgeRock defines Major, Minor, and Maintenance product release levels. The release level is reflected in the version number. The release level tells you what sort of compatibility changes to expect.

*Table A.1. Release Level Definitions*

Release Label	Version Numbers	Characteristics
Major	Version: x[.0.0] (trailing 0s are optional)	<ul style="list-style-type: none"> <li>• Bring major new features, minor features, and bug fixes</li> <li>• Can include changes even to Stable interfaces</li> <li>• Can remove previously Deprecated functionality, and in rare cases remove Evolving functionality that has not been explicitly Deprecated</li> <li>• Include changes present in previous Minor and Maintenance releases</li> </ul>
Minor	Version: x.y[.0] (trailing 0s are optional)	<ul style="list-style-type: none"> <li>• Bring minor features, and bug fixes</li> </ul>

Release Label	Version Numbers	Characteristics
		<ul style="list-style-type: none"> <li>• Can include backwards-compatible changes to Stable interfaces in the same Major release, and incompatible changes to Evolving interfaces</li> <li>• Can remove previously Deprecated functionality</li> <li>• Include changes present in previous Minor and Maintenance releases</li> </ul>
Maintenance	Version: x.y.z	<ul style="list-style-type: none"> <li>• Bring bug fixes</li> <li>• Are intended to be fully compatible with previous versions from the same Minor release</li> </ul>

## A.2. ForgeRock Product Interface Stability

ForgeRock products support many protocols, APIs, GUIs, and command-line interfaces. Some of these interfaces are standard and very stable. Others offer new functionality that is continuing to evolve.

ForgeRock acknowledges that you invest in these interfaces, and therefore must know when and how ForgeRock expects them to change. For that reason, ForgeRock defines interface stability labels and uses these definitions in ForgeRock products.

*Table A.2. Interface Stability Definitions*

Stability Label	Definition
Stable	This documented interface is expected to undergo backwards-compatible changes only for major releases. Changes may be announced at least one minor release before they take effect.
Evolving	<p>This documented interface is continuing to evolve and so is expected to change, potentially in backwards-incompatible ways even in a minor release. Changes are documented at the time of product release.</p> <p>While new protocols and APIs are still in the process of standardization, they are Evolving. This applies for example to recent Internet-Draft implementations, and also to newly developed functionality.</p>
Deprecated	This interface is deprecated and likely to be removed in a future release. For previously stable interfaces, the change was likely announced in a previous release. Deprecated interfaces will be removed from ForgeRock products.
Removed	This interface was deprecated in a previous release and has now been removed from the product.
Internal/Undocumented	Internal and undocumented interfaces can change without notice. If you depend on one of these interfaces, contact ForgeRock support or email <a href="mailto:info@forgerock.com">info@forgerock.com</a> to discuss your needs.

# Index

## A

- Account lockout, 66
- Authentication, 5
  - Active Directory, 7
  - Adaptive risk, 11
  - Anonymous, 17
  - Chains, 5, 60
  - Core, 22
  - Data Store, 29
  - Federation, 36
  - HMAC One-Time Password (HOTP), 37
  - HTTP Basic, 39
  - JDBC, 39
  - Kerberos, 58
  - LDAP, 41
  - Levels, 6
  - Membership (& self-registration), 44
  - Modules, 5
  - MSISDN, 45
  - OAuth 2.0, 49
  - Open Authentication (OATH), 47
  - RADIUS, 56
  - Secure Attribute Exchange (SAE), 57
  - SecurID, 57
  - Windows Desktop SSO, 58
  - Windows NT, 59
  - WSSAuth, 60
  - X509 certificate based, 18
- Authorization, 69, 78
  - Configuring, 70, 75, 79, 222, 240
  - Delegating, 77

## B

- Backup, 252

## C

- Caching, 270
- Certificates, 254
- Command line tools overview, 3
- Console overview, 1
- Cookie
  - Persistent, 55

- Cross-domain single sign on (CDSSO), 172

## D

- Dashboard service, 179, 181, 181
- Debug logging
  - Level, 263
  - Rotation, 264
  - Service selection, 264
  - Single file, 264
- Delegating administration, 82
- Device
  - Fingerprinting, 30

## E

- Enabling ssoadm.jsp, 4
- Entitlements, 78
  - Configuring, 79

## F

- Federation, 182
  - Changing signing key, 257
  - Configuring, 183, 184, 185, 185, 185, 191, 196, 201, 202, 203, 212
  - Linking accounts, 213
  - OAuth 2.0, 225
  - SAML 2.0 Single Logout (SLO), 206
  - SAML 2.0 Single Sign-On (SSO), 206

## G

- Google Apps, 202

## I

- Identity Gateway, 86

## L

- Logging, 263

## M

- Monitoring, 260
  - Health check, 263
  - JMX, 261
  - SNMP, 262

## O

- OAuth 2.0, 49, 149, 222

OpenID Connect 1.0, 149, 240

## **P**

Password reset, 160

Performance, 266

Policy, 69, 78

- Configuring, 70, 75, 79

- Delegating, 77, 83

Policy agents, 86

- Configuring, 90, 113, 139, 142, 145, 149, 149, 151

- Creating profiles, 87, 90

- Group inheritance, 88

- Profiles, 87

Post authentication plugins, 62

## **R**

Realms, 81

- Creating, 81

Restoring, 252

## **S**

Salesforce CRM, 203

Securing OpenAM, 278

Session quotas, 67

Sessions, 265

Silent installation, 3

Single Sign-On (SSO), 168

SSL, 254

## **T**

Troubleshooting, 284