For Review Purposes Only

Sun OpenSSO Enterprise 8.0 Technical Overview

Beta

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–3740–10
October 2008

080819@20795

List of Remarks

Contents

Preface

Sun OpenSSO Enterprise 8.0 is an access management product that includes a set of software components to provide the authentication and authorization services needed to support enterprise applications distributed across a network or Internet environment. This book, *Sun OpenSSO Enterprise 8.0 Technical Overview*, describes the features of OpenSSO Enterprise, explains what it does, and illustrates how it works.

Before You Read This Book

This book is intended for use by IT administrators and software developers who implement a web access platform using Sun servers and software. Readers of this guide should be familiar with the following technologies:

- SOAP
- Liberty Alliance Project specifications
- WS-* Specifications
- Security Assertion Markup Language (SAML) Specifications
- eXtensible Markup Language (XML)
- Lightweight Directory Access Protocol (LDAP)
- Java™
- JavaServer Pages™ (JSP)
- HyperText Transfer Protocol (HTTP)
- HyperText Markup Language (HTML)

Related Books

Related documentation is available as follows:

- "OpenSSO Enterprise 8.0 Core Documentation" on page 11
- "Adjunct Product Documentation" on page 12

OpenSSO Enterprise 8.0 Core Documentation

The OpenSSO Enterprise 8.0 core documentation set contains the following titles:

- The *Sun Federated Access Manager 8.0 Early Access (EA) Release Notes* will be available online after the product is released. It gathers an assortment of last-minute information, including a description of what is new in this current release, known problems and limitations, installation notes, and how to report issues with the software or the documentation.
- The *Sun OpenSSO Enterprise 8.0 Technical Overview* (this guide) provides an overview of how OpenSSO Enterprise components work together to protect enterprise assets and web-based applications. It also explains basic concepts and terminology.
- The *Sun OpenSSO Enterprise 8.0 Deployment Planning Guide* provides planning and deployment solutions for OpenSSO Enterprise based on the solution life cycle
- The *Sun OpenSSO Enterprise 8.0 Installation and Configuration Guide* provides information for installing and configuring OpenSSO Enterprise.
- The XXXXX provides information on how to tune Access Manager and its related components for optimal performance.
- The *Sun Federated Access Manager 8.0 Administration Guide* describes administrative tasks such as *how to create a realm* and *how to configure a policy*. Most of the tasks described can be performed using the administration console as well as the `famadm` command line utilities.
- The *Sun Federated Access Manager Administration Reference* is a look-up guide containing information about the command line interfaces, configuration attributes, internal files, and error codes.
- The *Sun Federated Access Manager 8.0 Developer's Guide* offers information on how to customize Access Manager and integrate its functionality into an organization's current technical infrastructure. It also contains details about the programmatic aspects of the product and its API.
- The *Sun Federated Access Manager 8.0 C API Reference* provides summaries of data types, structures, and functions that make up the public Access Manager C APIs.
- The *Federated Access Manager 8.0 Java API Reference* provides information about the implementation of Java packages in Access Manager.
- The *Sun Java System Federated Access Manager Policy Agent 3.0 User's Guide* provides an overview of the policy functionality and the policy agents available for OpenSSO Enterprise.

Updates to the *Release Notes* and links to modifications of the core documentation can be found on the OpenSSO Enterprise page at docs.sun.com. Updated documents will be marked with a revision date.

Adjunct Product Documentation

Useful information can be found in the documentation for the following products:

- Sun Java System Directory Server Enterprise Edition 6.0
- Sun Java System Web Server 7.0

- Sun Java System Application Server Enterprise Edition 8.2
- Sun Java System Web Proxy Server 4.0.4

Searching Sun Product Documentation

Besides searching Sun product documentation from the docs.sun.com℠ web site, you can use a search engine by typing the following syntax in the search field:

search-term `site:docs.sun.com`

For example, to search for "broker," type the following:

```
broker site:docs.sun.com
```

To include other Sun web sites in your search (for example, java.sun.com, www.sun.com, and developers.sun.com), use `sun.com` in place of `docs.sun.com` in the search field.

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- Documentation (`http://www.sun.com/documentation/`)
- Support (`http://www.sun.com/support/`)
- Training (`http://www.sun.com/training/`)

Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. To share your comments, go to `http://docs.sun.com` and click Send Comments. In the online form, provide the full document title and part number. The part number is a 7-digit or 9-digit number that can be found on the book's title page or in the document's URL. For example, the title of this book is *Sun OpenSSO Enterprise 8.0 Technical Overview*, and the part number is 820–3740.

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P–1 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
AaBbCc123	A placeholder to be replaced with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized (note that some emphasized items appear bold online)	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file.

Shell Prompts in Command Examples

The following table shows default system prompts and superuser prompts.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell on UNIX and Linux systems	`machine_name%`
C shell superuser on UNIX and Linux systems	`machine_name#`

TABLE P–2 Shell Prompts *(Continued)*

Shell	Prompt
Bourne shell and Korn shell on UNIX and Linux systems	`$`
Bourne shell and Korn shell superuser on UNIX and Linux systems	`#`
Microsoft Windows command line	`C:\`

Symbol Conventions

The following table explains symbols that might be used in this book.

TABLE P–3 Symbol Conventions

Symbol	Description	Example	Meaning
`[ ]`	Contains optional arguments and command options.	`ls [-l]`	The `-l` option is not required.
`{ | }`	Contains a set of choices for a required command option.	`-d {y|n}`	The `-d` option requires that you use either the `y` argument or the `n` argument.
`${ }`	Indicates a variable reference.	`${com.sun.javaRoot}`	References the value of the `com.sun.javaRoot` variable.
-	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.
+	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.
→	Indicates menu item selection in a graphical user interface.	File → New → Templates	From the File menu, choose New. From the New submenu, choose Templates.

PART I

An Overview of Sun OpenSSO Enterprise 8.0

This part of the Sun OpenSSO Enterprise 8.0 Technical Overview contains introductory material concerning Sun OpenSSO Enterprise 8.0 (OpenSSO Enterprise). It includes the following chapters:

◆ ◆ ◆ CHAPTER 1

Introducing OpenSSO Enterprise

Sun OpenSSO Enterprise 8.0 (OpenSSO Enterprise) integrates authentication and authorization services, single sign-on (SSO), and open, standards-based federation protocols (including the Liberty Alliance Project specifications, WS-Federation and Security Assertion Markup Language [SAML]) to provide a comprehensive solution for protecting network resources by preventing unauthorized access to web services, applications and web content, and securing identity data. This introductory chapter contains a high-level description of OpenSSO Enterprise and what it does. It contains the following sections:

What is OpenSSO Enterprise?

OpenSSO Enterprise is a single product that combines the features of Sun Java™ System Access Manager, Sun Java System Federation Manager, and the Sun Java System SAML v2 Plug-in for Federation Services. Additionally, it is enhanced with new functionality developed specifically for this release. OpenSSO Enterprise provides access management by allowing the implementation of authentication, policy-based authorization, federation, SSO, and web services security from a single, unified framework. The core application is delivered as a simple web archive (WAR) that can be easily deployed in a supported web container.

Note – OpenSSO Enterprise is Sun Microsystems' commercial distribution of the open source code available at OpenSSO.

To assist the core application, policy agents, the Client SDK, and (possibly) other disparate pieces must be installed remotely and be able to communicate with the OpenSSO Enterprise

server. See "What Does OpenSSO Enterprise Do?" on page 20 for a high-level picture of the deployment architecture and Chapter 2, "Examining OpenSSO Enterprise," for more specific information.

What Does OpenSSO Enterprise Do?

The following types of interactions occur daily in a corporate environment.

- An employee looks up a colleague's phone number in the corporate phone directory.
- A manager retrieves employee salary histories to determine an individual's merit raise.
- An administrative assistant adds a new hire to the corporate database, triggering the company's health insurance provider to add the new hire to its enrollment.
- An engineer sends an internal URL for a specification document to another engineer who works for a partner company.
- A customer logs into a company's web site and looks for a product in their online catalog.
- A vendor submits an invoice to the company's accounting department.
- A corporate human resources administrator accesses an outsourced benefits application.

For each of these transactions, the company must determine who is allowed to view the information or use the application. Some information such as product descriptions and advertising can be made available to everyone in a public online catalog. Other information such as accounting and human resources data must be restricted to employees only. And other sensitive information such as pricing models and employee insurance plans is appropriate to share only with partners, suppliers, and employees. This need for access determination is met by Sun OpenSSO Enterprise, an access management product with authentication, authorization, and single sign-on (SSO) services provided out of the box.

When a user or an external application requests access to content stored on a company's server, a *policy agent* (available in a separate download and installed on the same machine as the resource you want to protect) intercepts the request and directs it to OpenSSO Enterprise which, in turn, requests credentials (such as a username and password in the case of a user) for authentication. If the credentials returned match those stored in the appropriate identity data store, OpenSSO Enterprise determines that the user is authentic. Following authentication, access to the requested content is determined by the policy agent which evaluates the policies associated with the authenticated identity. Policies are created using OpenSSO Enterprise and identify which identities are allowed to access a particular resource, specifying the conditions under which this authorization is valid. Based upon the results of the policy evaluation, the policy agent either grants or denies the user access. Figure 1–1 illustrates a high-level deployment architecture of OpenSSO Enterprise.

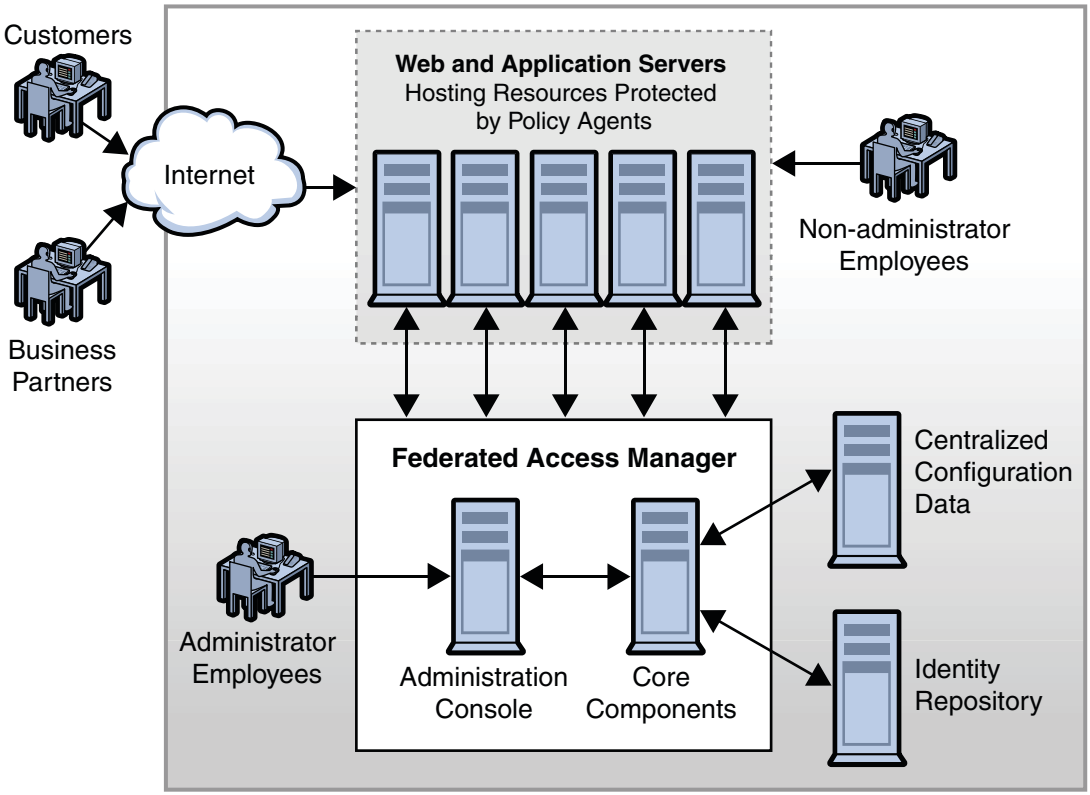


FIGURE 1–1 High-level Deployment Architecture of OpenSSO Enterprise

What Are the Functions of OpenSSO Enterprise?

The following sections contain an overview of the functions of OpenSSO Enterprise.

Access Control

OpenSSO Enterprise manages authorized access to network services and resources. By implementing authentication and authorization, OpenSSO Enterprise (along with an installed policy agent) ensures that access to protected resources is restricted to authorized users. In a nutshell, a policy agent intercepts a request for access to a resource and communicates with OpenSSO Enterprise to authenticate the requestor. If the user is successfully authenticated, the

policy agent then evaluates the policies associated with the requested resource and the user to determine if the authenticated user is authorized to access the resource. If the user is authorized, the policy agent allows access to the resource, also providing identity data to the resource to personalize the interaction. For more information on access control, see "Core Services" on page 31 and Part II.

Federation Management

With the introduction of federation protocols into the process of access management, identity information and entitlements can be communicated across security domains, spanning multiple trusted partners. By configuring a *circle of trust* and defining applications and services as *providers* in the circle (either identity providers or service providers), users can opt to associate, connect or bind the various identities they have configured locally for these providers. The linked local identities are federated and allow the user to log in to one identity provider site and click through to an affiliated service provider site without having to reauthenticate; in effect, single sign-on (SSO). OpenSSO Enterprise supports several open federation technologies including the Security Access Markup Language (SAML) versions 1 and 2, WS-Federation, and the Liberty Alliance Project Identity Federation Framework (Liberty ID-FF), therefore encouraging an interoperable infrastructure among providers. For more information on federation management, see "Core Services" on page 31 and Part III.

Web Services Security

A *web service* is a component service or application that exposes some type of business or infrastructure functionality through a language-neutral and platform-independent, network interface; enterprises might use this web service to build larger service-oriented architectures. In particular, the service defines its interface (for example, the format of the message being exchanged) using the Web Services Description Language (WSDL), and communicates using SOAP and eXtensible Markup Language (XML) messages. The web service client (WSC) communicates with the web service provider (WSP) through an intermediary — usually a firewall or load balancer.

Although web services enable open, flexible, and adaptive interfaces, their openness creates security risks. Without proper security protections, a web service can expose vulnerabilities that might have dire consequences. Hence, ensuring the integrity, confidentiality and security of web services through the application of a comprehensive security model is critical for both enterprises and consumers. A successful security model associates identity data with the web services and creates secure service-to-service interactions. The security model adopted by OpenSSO Enterprise identifies the user and preserves that identity through multiple interactions, maintains privacy and data integrity, uses existing technologies, and logs the interactions. In OpenSSO Enterprise, the following web service security standards are implemented:

- Liberty Alliance Project Identity Web Services Framework (Liberty ID-WSF)

- WS-I Basic Security Profile
- WS-Trust (from which the Security Token Service was developed)

For more information on OpenSSO Enterprise web services and web services security, see "Core Services" on page 31 and Part IV.

Identity Web Services

For some time, OpenSSO Enterprise has provided client interfaces for access to core features and functionality. These interfaces are used by policy agents and custom applications developed by customers. With this release, OpenSSO Enterprise now exposes certain functions as simple identity web services allowing developers to easily invoke them when developing their applications using one of the supported integrated development environment (IDE) products. (The IDE generates the stub code that wraps a call to the web service.) Identity Web Services are available using:

- SOAP and Web Services Description Language (WSDL)
- Representational State Transfer (REST)

They do not require the deployment of an agent or a proxy and include the following capabilities:

- Authentication to validate user credentials.
- Authorization to permit access to protected resources.
- Provisioning for user attribute management and self registration.
- Logging to keep track of it all.

For more information on identity services, see "Identity Web Services" on page 48 and Part IV.

What Else Does OpenSSO Enterprise Offer?

OpenSSO Enterprise allows for:

- **Ease of Deployment:** OpenSSO Enterprise is delivered as a web archive (WAR) that can be easily deployed as a Java EE application in different web containers. Most configuration files and required libraries are inside the WAR to avoid the manipulation of the classpath in the web container's configuration file. The OpenSSO Enterprise WAR is supported on:
  1. Sun Java System Web Server 7.0 — Update 3 and above
  2. Sun Java System Application Server 9.1 EE Update 2 and above (and Glassfish v2 update 2 and above)
  3. BEA WebLogic Application Server 9.2 mp2
  4. IBM WebSphere Application Server 6.1
  5. Oracle Application Server 10g

6. JBoss 4.2.x
7. **[Remark 1–1 Reviewer: more details to come from Indira per review]** Tomcat 5.5.x & 6.x
8. Geronimo (supported on the Sun Solaris™ 10 Operating Environment for SPARC, x86 & x64 and the Sun Solaris 9 Operating Environment for SPARC & x86 systems only)

Note – Geronimo can install Tomcat and Jetty web containers; OpenSSO Enterprise supports only Tomcat.

See the *Sun Federated Access Manager 8.0 Early Access (EA) Release Notes* for updates to this list.

- **Portability:** OpenSSO Enterprise is supported on the following operating systems:
  1. Sun Solaris 10 Operating Environment for SPARC, x86 & x64 systems
  2. Sun Solaris 9 Operating Environment for SPARC & x86 systems
  3. Windows Server 2003 and Windows XP (development only) operating systems
  4. Red Hat Enterprise Linux 4 Server (Base)
  5. Red Hat Enterprise Linux 4 Advanced Platform
  6. Red Hat Enterprise Linux 5 Server (Base)
  7. Red Hat Enterprise Linux 5 Advanced Platform
  8. Windows 2003 Standard Server
  9. Windows 2003 Enterprise Server
  10. Windows 2003 Datacenter Server
  11. Windows Vista
  12. IBM AIX 5.3 (supported with the IBM WebSphere Application Server 6.1 container only)

  See the *Sun Federated Access Manager 8.0 Early Access (EA) Release Notes* for updates to this list.
- **Open Standards:** OpenSSO Enterprise is built using open standards and specifications as far as possible. For example, features designed for federation management and web services security are based on the Security Assertion Markup Language (SAML), the Liberty Alliance Project specifications, and the WS-Security standards.
- **Ease of Administration:** OpenSSO Enterprise contains a web-based, graphical administration console as well as command line interfaces for configuration tasks and administrative operations. Additionally, an embedded, centralized data store allows for one place to store server and agent configuration data.
- **Security:**

1. Runtime security enables an enterprise's resources to be protected as configured and OpenSSO Enterprise services to be accessed by authorized entities only.
2. Administration security ensures only authorized updates are made to the OpenSSO Enterprise configuration data.
3. Deployment security implements best practices for installing OpenSSO Enterprise on different operating systems, web containers, and so forth.

Additionally, all security actions are logged.

- **Configuration Data Store:** OpenSSO Enterprise can write server configuration data to an embedded configuration data store. You can also point to instances of Sun Java System Directory Server 5.2 or Directory Server Enterprise Edition 6.x during configuration of OpenSSO Enterprise for use as a configuration data store. See "Data and Data Stores" on page 54 for more information.
- **User Data Store Independence:** OpenSSO Enterprise allows you to view and retrieve user information without making changes to an existing user database. Supported directory servers include Directory Server 5.1, 5.2 & 6.2, IBM Tivoli Directory 6.1, and Microsoft Active Directory 2003. See "Data and Data Stores" on page 54 for more information.

Caution – The configuration data store embedded with OpenSSO Enterprise should only be used as a user data store for proof of concepts and deployments in development.

- **Web and Non-Web-Based Resources:** The core design of OpenSSO Enterprise caters to SSO for both web and non-web applications.
- **Performance, Scalability and Availability:** OpenSSO Enterprise can be scaled horizontally and vertically to handle increased workloads, and as security needs change over time. There is no single point of failure.
- **Distributed Architecture** Server and client components can be deployed across the enterprise or across domain boundaries as all application programming interfaces (API) provide remote access to OpenSSO Enterprise based on a service-oriented architecture.
- **Flexibility and Extensibility:** Many OpenSSO Enterprise services expose a service provider interface (SPI) allowing expansion of the framework to provide for specific deployment needs.
- **Internationalization** OpenSSO Enterprise contains a framework for multiple language support. Customer facing messages, API, command line interfaces, and user interfaces are localized in the supported languages.

◆ ◆ ◆ CHAPTER 2

Examining OpenSSO Enterprise

OpenSSO Enterprise provides a pluggable architecture to deliver access management, secure web services, and federation capabilities. This chapter contains information on the internal architecture and features of OpenSSO Enterprise.

OpenSSO Enterprise Client/Server Architecture

OpenSSO Enterprise is written in Java, and leverages many industry standards, including the HyperText Transfer Protocol (HTTP), the eXtensible Markup Language (XML), the Security Assertion Markup Language (SAML), and SOAP, to deliver access management, secure web services, and federation capabilities in a single deployment. It consists of client application programming interfaces (a Client Software Development Kit [SDK]), a framework of services that implement the business logic, and service provider interfaces (SPI) that are implemented by concrete classes and can be used to extend the functionality of OpenSSO Enterprise as well as retrieve information from data stores. Figure 2–1 illustrates the client/server architecture of OpenSSO Enterprise.

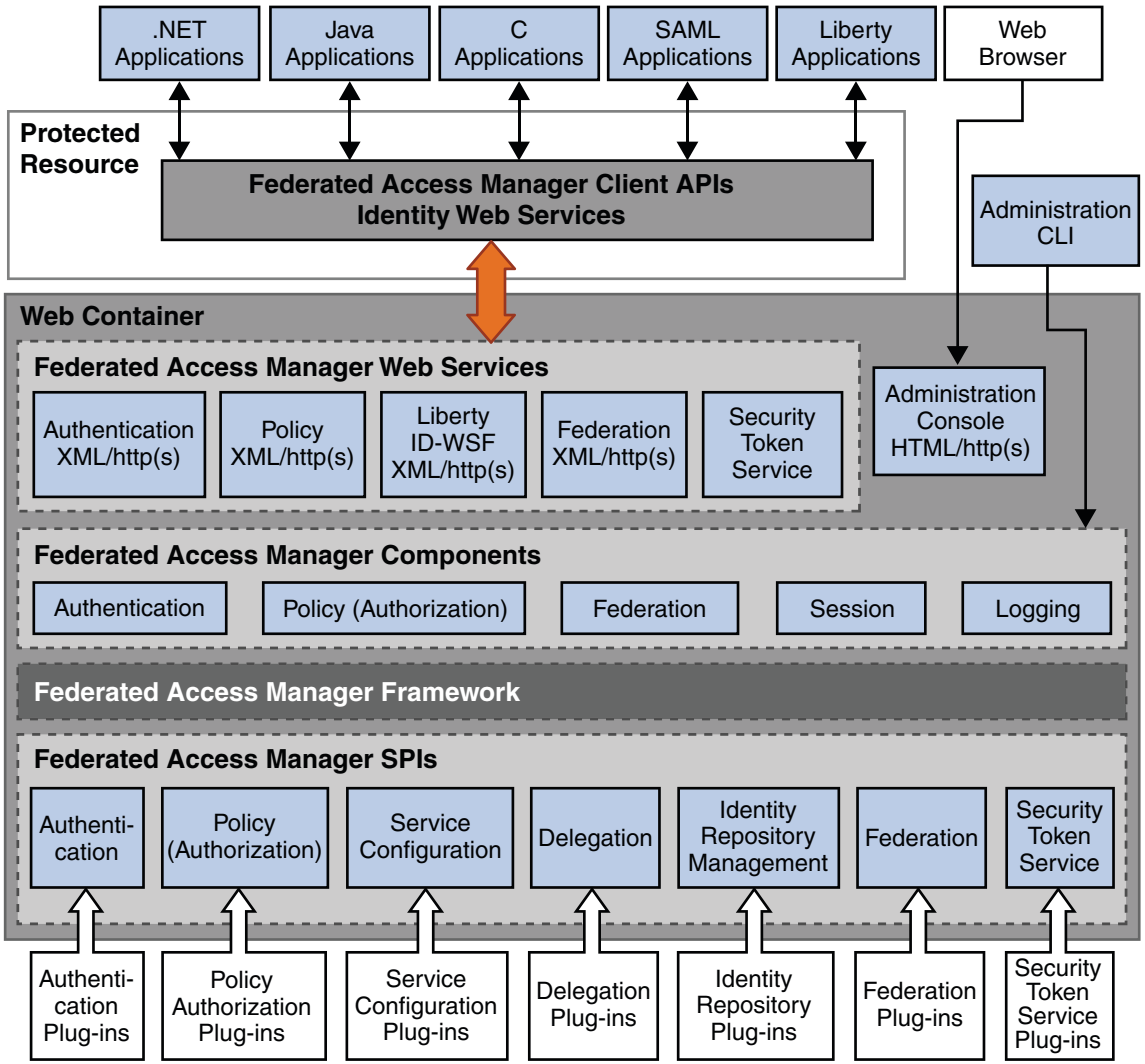


FIGURE 2–1 Client/Server Architecture of OpenSSO Enterprise

Each component of OpenSSO Enterprise uses its own framework to retrieve customer data from the plug-in layer and to provide data to other components. The OpenSSO Enterprise framework integrates all of the application logic into one layer that is accessible to all components and plug-ins. The Client SDK and Identity Web Services are installed on a machine remote to the OpenSSO Enterprise server that holds a resource to be protected; they provide remote access to the OpenSSO Enterprise for client applications. (The policy agent, also installed on the remote machine, is basically a client written using the Client SDK and Identity Web Services.) Applications on the remote machine access OpenSSO Enterprise using the

Client SDK. Custom plug-in modules are installed on the machine local to OpenSSO Enterprise and interact with the OpenSSO Enterprise SPI to retrieve required information from the appropriate data store and deliver it to the plug-ins and, in turn, the OpenSSO Enterprise framework for processing.

How OpenSSO Enterprise Works

To gain access to a protected resource, the requestor needs to be authenticated and have the authorization to access the resource. When someone (using a browser) sends an HTTP request for access to a protected resource, a policy agent (separately downloaded and installed on the same machine as the resource you want to protect) intercepts the request and examines it. If no valid OpenSSO Enterprise *session token* (to provide proof of authentication) is found, the policy agent contacts the server which then invokes the authentication and authorization processes. Figure 2–2 illustrates one way in which the policy agents can be situated to protect an enterprise's servers by directing HTTP requests to a centralized OpenSSO Enterprise for processing.

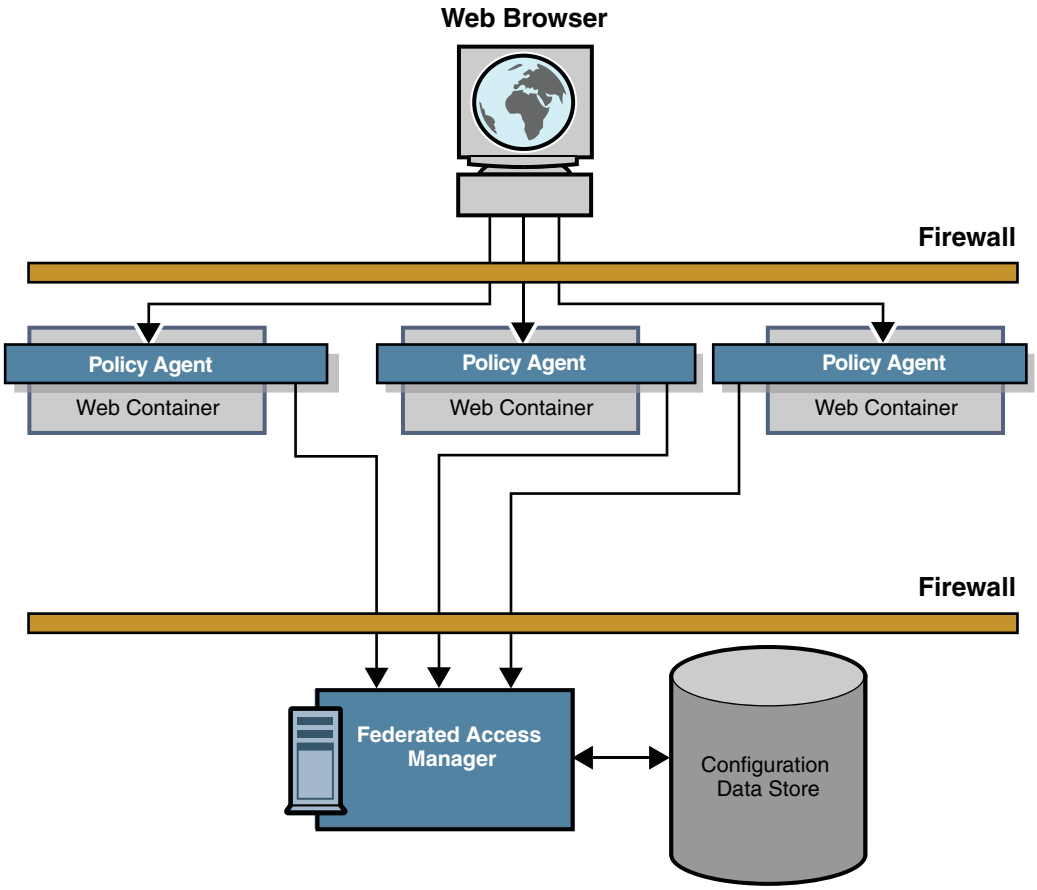


FIGURE 2–2 Basic OpenSSO Enterprise Deployment

OpenSSO Enterprise integrates core features such as access control through authentication and authorization processes, and federation. These functions can be configured using the administration console or the `famadm` command line utility. Figure 2–3 is a high-level illustration of the interactions that occur between parties (including the policy agent, browser, and protected application) during authentication and authorization in a OpenSSO Enterprise deployment.

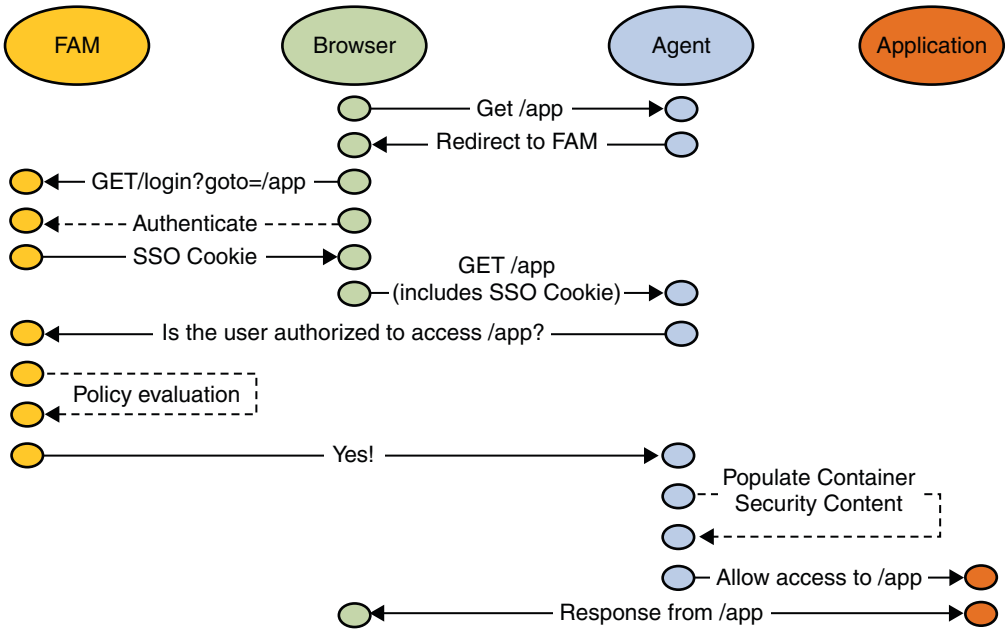


FIGURE 2–3 OpenSSO Enterprise Authentication and Authorization Interactions

For more information, see Part II.

Core Services

Services developed for OpenSSO Enterprise generally contain both a server component and a client component. The server component is a simple Java servlet developed to receive XML requests and return XML responses. (The deployment descriptor `web.xml` defines the servlet name and description, the servlet class, initialization parameters, mappings, and other startup information.) The client component is provided as Java application programming interfaces (API), and in some cases C API, that allow remote applications and other OpenSSO Enterprise services to communicate with and consume the particular functionality.

Each core service uses its own framework to retrieve customer and service data and to provide it to other OpenSSO Enterprise services. The OpenSSO Enterprise framework integrates all of these service frameworks to form a layer that is accessible to all product components and plug-ins. The following sections contain information on the OpenSSO Enterprise core services.

- "Federation Services" on page 44
- "Web Services Stack" on page 46
- "Web Services Security" on page 47
- "Identity Web Services" on page 48

Note – Many services also provide a public SPI that allows the service to be extended. See the *Sun Federated Access Manager 8.0 Developer's Guide*, the *Sun Federated Access Manager 8.0 C API Reference*, and the *Federated Access Manager 8.0 Java API Reference* for information.

Authentication Service

The Authentication Service provides the functionality to request user credentials and validate them against a specified authentication data store. Upon successful authentication, it creates a session data structure for the user that can be validated across all web applications participating in an SSO environment. Several authentication modules are supplied with OpenSSO Enterprise, and new modules can be plugged-in using the Java Authentication and Authorization Service (JAAS) SPI.

Note – The Authentication Service is based on the JAAS specification, a set of API that enables services to authenticate and enforce access controls upon users. See the Java Authentication and Authorization Service Reference Guide for more information.

Components of the Authentication Service include:

- The Distributed Authentication User Interface allows the Authentication Service user interface to be deployed separately from OpenSSO Enterprise, if desired. By deploying this authentication proxy in the DMZ and using the authentication interfaces provided in the Client SDK to pass user credentials back and forth, you can protect OpenSSO Enterprise data (for example, the login URL information and hence the host information). JavaServer Pages (JSP) represent the interface displayed to users for authentication and are completely customizable.
- The Core Authentication Service executes common processes across all authentication modules. Key responsibilities of this service include identification of the appropriate plan to authenticate the user (identify the authentication module, load the appropriate JSP) and creation of the appropriate session for the authenticated user.
- The Authentication API are *remoteable* interfaces that don't need to reside on the same machine as the OpenSSO Enterprise server. This allows remote clients to access the Authentication Service. remote-auth.dtd defines the structure for the XML communications that will be used by the Authentication Service, providing definitions to initiate the process, collect credentials and perform authentication.

- A number of authentication modules are installed and configured (including, but not limited to, LDAP, RADIUS, Windows Desktop, Certificate, and Active Directory). A configured authentication level for each module is globally defined. Mechanisms are also provided to upgrade a user's session after authenticating the user to an additional authentication module that satisfies the authentication level of the resource. New modules can be plugged-in using the JAAS SPI.

The Authentication Service interacts with both the database that stores user credentials (authentication data store) to validate the user, and with the Identity Repository Service plug-ins to retrieve user profile attributes. When the Authentication Service determines that a user's credentials are genuine, a valid user session token is issued, and the user is said to be *authenticated*. Figure 2–4 illustrates how the local and remote authentication components interact within a OpenSSO Enterprise deployment.

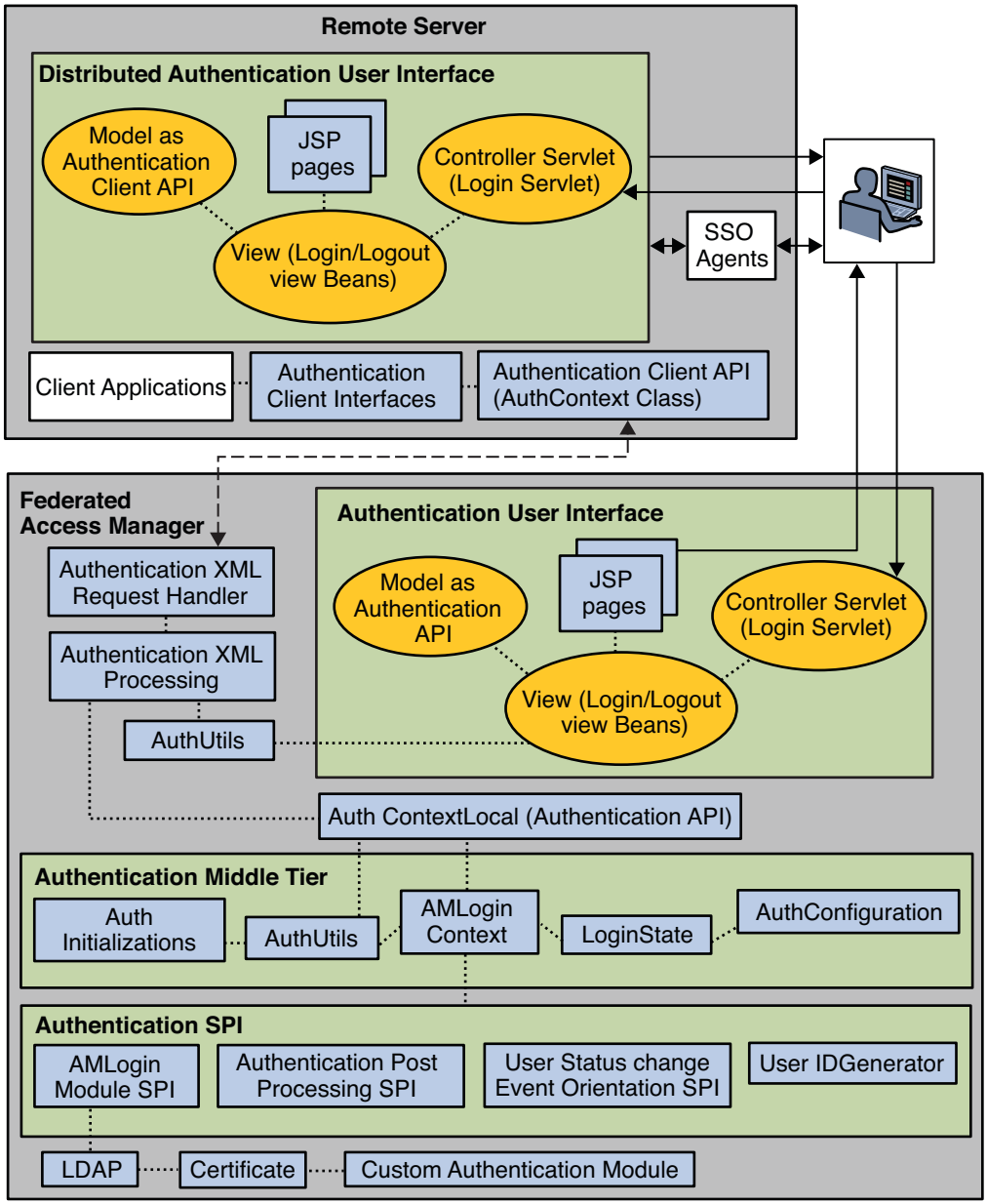


FIGURE 2–4 Authentication Service Components Within a OpenSSO Enterprise Deployment

More information on the architecture of the Authentication Service can be found in the Authentication Service Architecture document on the OpenSSO web site.

Policy Service

Authorization is the process with which OpenSSO Enterprise evaluates the policies associated with an authenticated user's identity, and determines whether the user has permission to access a protected resource. (A *policy* defines the rules that specify a user's access privileges to a protected resource.) The Policy Service provides the authorization functionality using a rules-based engine. It interacts with the OpenSSO Enterprise configuration data store, a delegation plug-in (which helps to determine the administrator's scope of privileges), and Identity Repository Service plug-ins to verify that the user has access privileges from a recognized authority. Policy can be configured using the administration console, and comprises the following:

- A *Schema* for the policy type (normal or referral) that describes the syntax of policy.
- A *Rule* which defines the policy itself and is made up of a *Resource*, an *Action* and a *Value*.
- *Condition(s)* to define constraints on the policy.
- *Subject(s)* to define the user or collection of users which the policy affects.
- A *ResponseProvider(s)* to send requested attribute values, typically based on the user profile, with the policy decision.

Figure 2–5 illustrates how the local and remote components of the Policy Service interact within a OpenSSO Enterprise deployment. Note that the `PolicyServiceRequestHandler` maps to the `PolicyRequest` XML element.

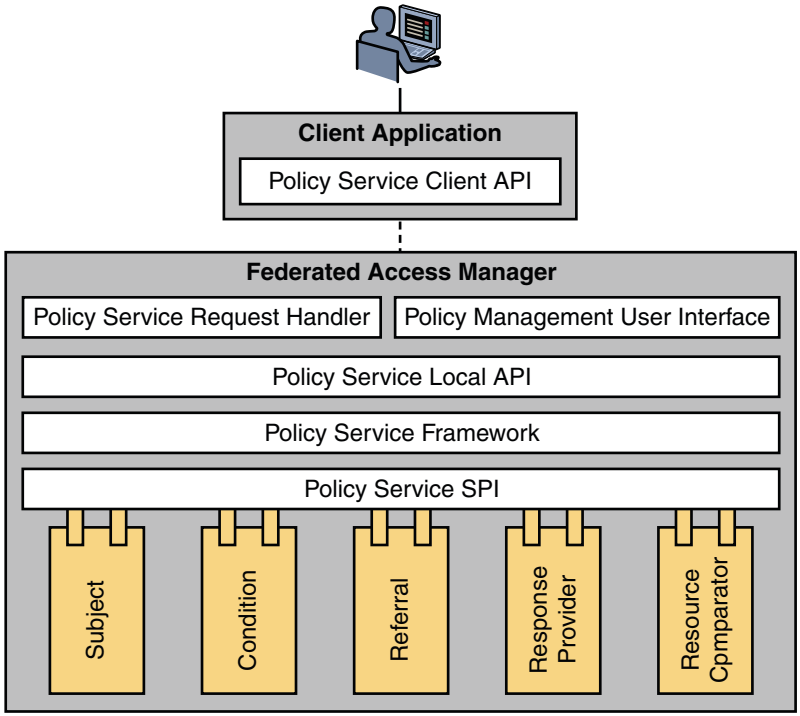


FIGURE 2–5 Policy Service Components within a OpenSSO Enterprise Deployment

Policy agents are an integral part of authorization. They are programs, available for installation separate from OpenSSO Enterprise, that police the web container which hosts the protected resources. When a user requests access to the protected resource (such as a server or an application), the policy agent intercepts the request and redirects it to the OpenSSO Enterprise Authentication Service. Following authentication, the policy agent will enforce the authenticated user's assigned policies. OpenSSO Enterprise supports two types of policy agents:

- The *web agent* is written in C and can protect any URL-based resource.
- The *Java Platform, Enterprise Edition (Java EE) agent* enforces URL-based policy and Java EE-based policy for Java applications on Java EE containers.

Note – When policy agents are implemented, all HTTP requests are implicitly denied unless explicitly allowed by the presence of two things:

1. A valid session
2. A policy allowing access

Note – If the resource is in the Not Enforced list defined for the policy agent, access is allowed even if there is no valid session.

More information on the architecture of the Policy Service can be found in the Policy Service Architecture document on the OpenSSO web site. For an overview of the available policy agents and links to specific information on installation, see the *Sun Java System Federated Access Manager Policy Agent 3.0 User's Guide*.

Session Service

The mission of the OpenSSO Enterprise Session Service is to maintain information about an authenticated user's session across all web applications participating in a user session. Additionally, OpenSSO Enterprise provides continuous proof of the user's identity, enabling the user to access multiple enterprise resources without having to provide credentials each time. This enables the following types of user sessions.

- **Basic user session.** The user provides credentials to log in to one application, and then logs out of the same application.
- **SSO session.** The user provides credentials once, and then accesses multiple applications within the same DNS domain.
- **Cross domain SSO (CDSSO) session.** The user provides credentials once, and then accesses applications among multiple DNS domains.

A *user session* is the interval between the time a user attempts authentication through OpenSSO Enterprise and is issued a session token, and the moment the session expires, is terminated by an administrator, or the user logs out. In what might be considered a typical user session, an employee accesses the corporate benefits administration service. The service, monitored by OpenSSO Enterprise, prompts the user for a username and password. With the credentials OpenSSO Enterprise can *authenticate*, or verify that the user is who he says he is. Following authentication, OpenSSO Enterprise allows the user access to the service providing authorization is affirmed. Successful authentication through OpenSSO Enterprise results in the creation of a *session data structure* for the user or entity by the Session Service. Generally speaking, the Session Service performs some or all of the following:

- Generates unique session identifiers, one for each user's session data structure

Note – A session data structure is initially created in the `INVALID` state with default values for certain attributes and an empty property list. Once the session is authenticated, the session state is changed to `VALID` and session data is updated with the user's identity attributes and properties.

- Maintains a master copy of session state information

Note – The session state maintained on the client side is a cached view of the actual session data structure. This cache can be updated by either the active polling mechanism or the session notification triggered by the Session Service.

- Implements time-dependent behavior of sessions — for example, enforces timeout limits
- Implements session life cycle events such as logout and session destruction
- Notifies all participants in the same SSO environment of session state changes
- Enables SSO and cross-domain single sign-on (CDSSO) among applications external to OpenSSO Enterprise by providing continued proof of identity.
- Allows participating clients to share information across deployments
- Implements high availability facilities

Figure 2–6 illustrates the interactions between the local and remote components of the Session Service within a OpenSSO Enterprise deployment.

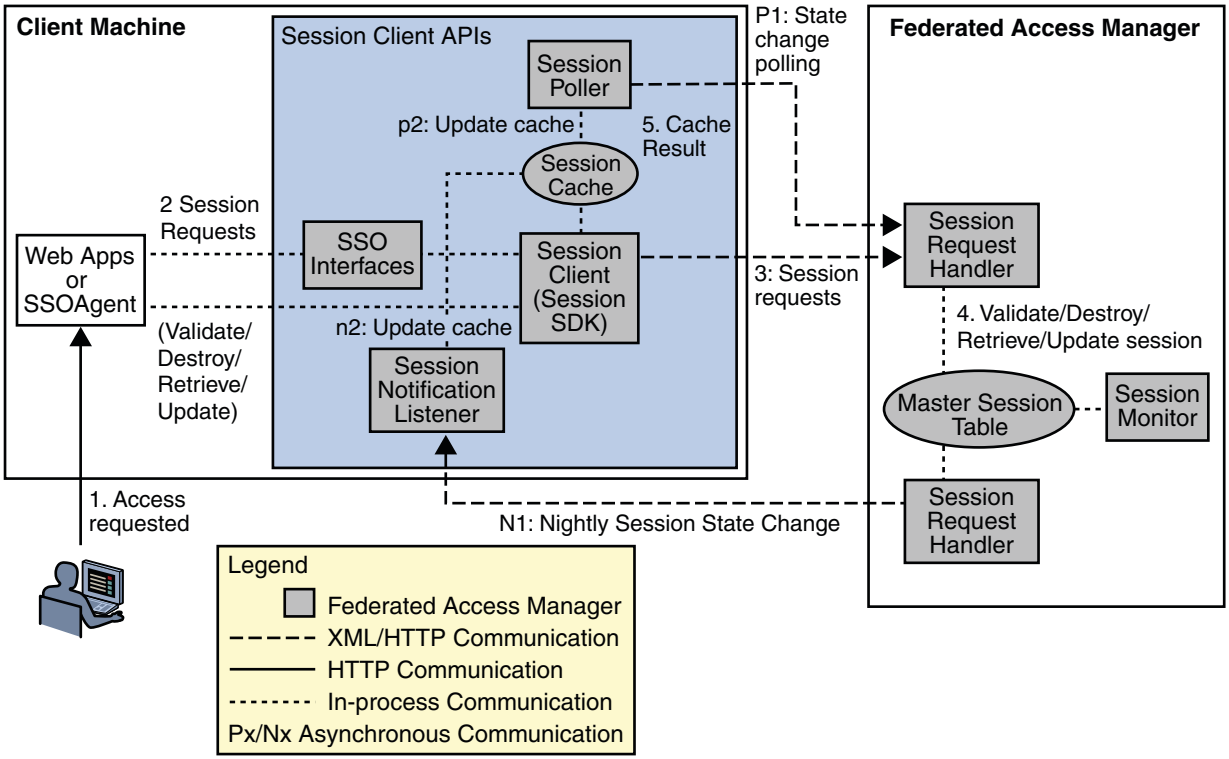


FIGURE 2–6 Session Service Components within a OpenSSO Enterprise Deployment

Additionally, Figure 2–7 illustrates how the messaging capabilities of Message Queue can be used to push session information to a persistent store based on the Berkeley DataBase (DB). Using OpenSSO Enterprise in this manner enables the following key feature:

- Session Failover allows an alternative OpenSSO Enterprise server to pick up a given user session when the server owning the original session fails.
- Session Constraints allow deployments to specify constraints on a sessions, such as one session per user.

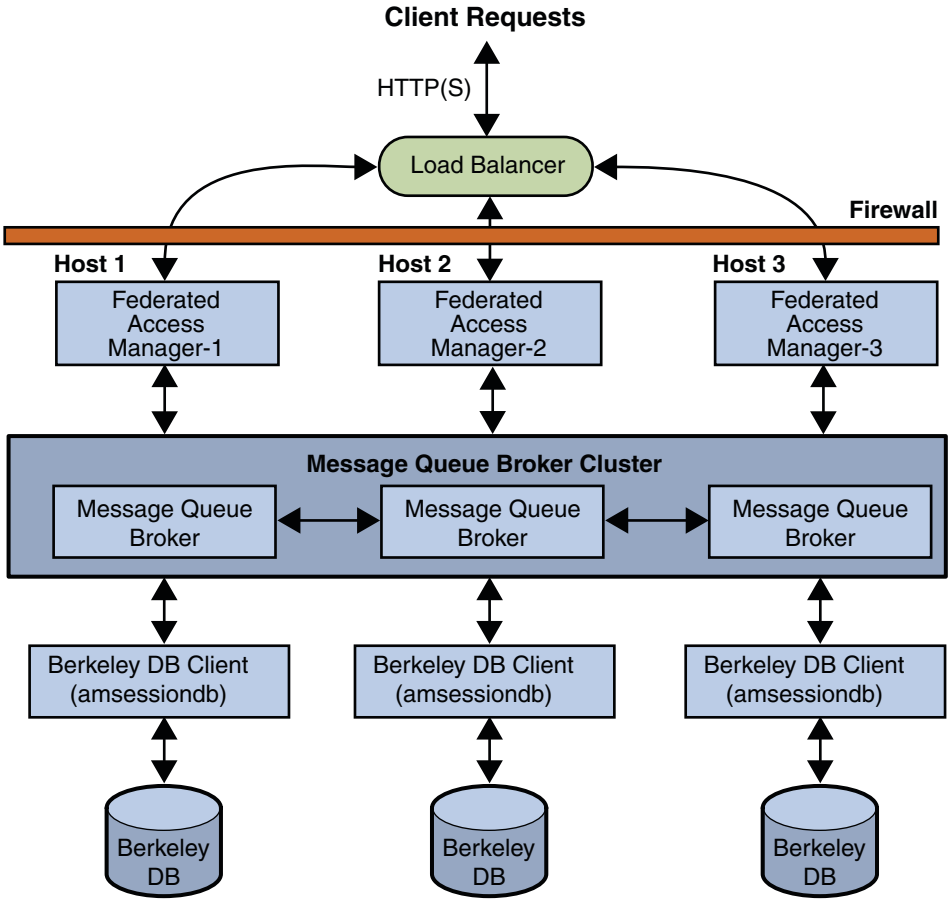


FIGURE 2–7 Session Persistence Deployment Architecture

More information on the architecture of the Session Service can be found in the Session Service Architecture document on the OpenSSO web site. For more information on session failover, see Chapter 5, "Implementing OpenSSO Enterprise Session Failover," in *Sun OpenSSO Enterprise 8.0 Installation and Configuration Guide*.

Logging Service

When a user logs in to a resource protected by OpenSSO Enterprise, the Logging Service records information about the user's activity. The common Logging Service can be invoked by components residing on the same server as OpenSSO Enterprise as well as those on the client machine, allowing the actual mechanism of logging (such as destination and formatting) to be separated from the contents which are specific to each component. You can write custom log operations and customize log plug-ins to generate log reports for specific auditing purposes.

Administrators can control log levels, authorize the entities that are allowed to create log entries and configure secure logging (the latter for flat files only). Logged information includes the name of the host, an IP address, the identity of the creator of the log entry, the activity itself, and the like. Currently, the fields logged as a *log record* are controlled by the Configurable Log Fields selected in the Logging Configuration page located under the System tab of the OpenSSO Enterprise console. The Logging Service is dependent on the client application (using the Logging APIs) creating a programmatic `LogRecord` to provide the values for the log record fields. The logging interface sends the logging record to the Logging Service which determines the location for the log record from the configuration. A list of active logs can also be retrieved using the Logging API. Figure 2–8 illustrates the interactions between the local and remote components of the Logging Service in a OpenSSO Enterprise deployment.

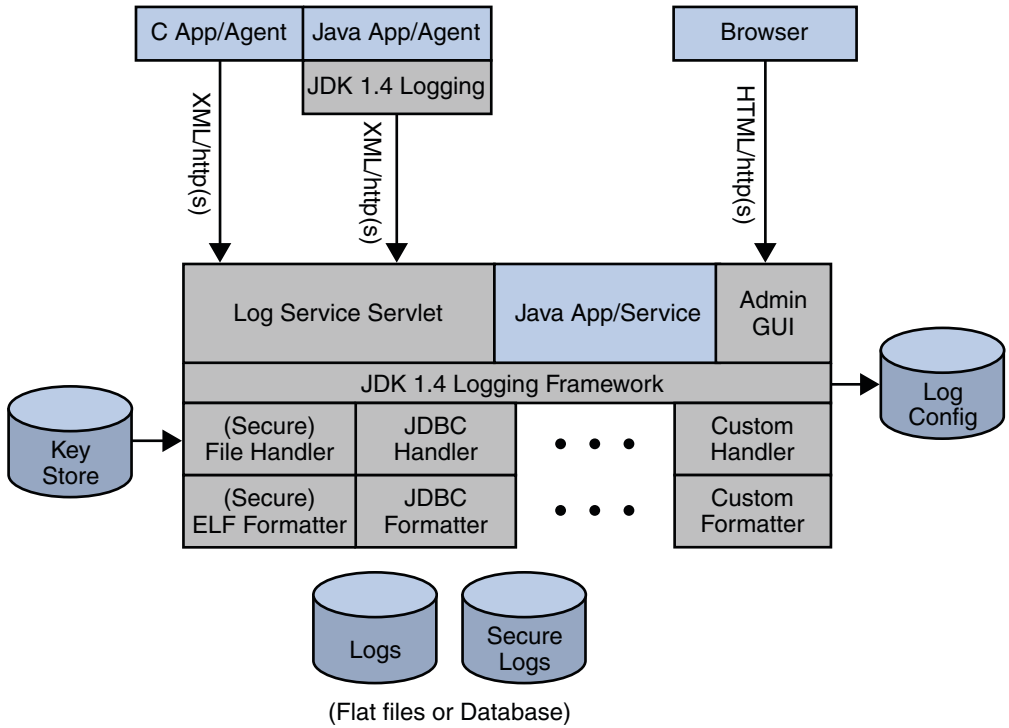


FIGURE 2–8 Logging Service Components within a OpenSSO Enterprise Deployment

Caution – Generally speaking, writing log records can be done remotely, using the Client SDK, but anything involving the reading API can only be done on the machine on which OpenSSO Enterprise is installed. Using the reading API uses a great deal of system resources, especially when database logging is involved.

Logs can be written to flat files or to one of the supported databases (Oracle and MySQL). See Chapter 5, "Recording Events with the Logging Service," for more information.

Identity Repository Service

The Identity Repository Service allows OpenSSO Enterprise to integrate an existing user data store (such as a corporate LDAP server) into the environment. The Identity Repository Service is able to access user profiles (as well as group and role assignments if supported) and is capable of spanning multiple repositories — even of different types. The Identity Repository Service is configured per realm under the Data Stores tab and its main functions are:

- To specify an identity repository that will store service configurations and attributes for users, groups and roles.
- To provide a list of identity repositories that can provide user attributes to the Policy Service and Federation Services frameworks.
- To combine the attributes obtained from different repositories.
- To provide interfaces to create, read, edit, and delete identity objects such as a realm, role, group, user, and agent.
- To map identity attributes using the Principal Name from the `SSOToken` object.

Access to the Identity Repository Service is provided by the `com.sun.identity.idm` Java package. The `AMIdentityRepository` class represents a realm that has one or more identity repositories configured and provides interfaces for searching, creating and deleting identities. The `AMIdentity` class represents an individual identity such as a user, group or role and provides interfaces to set, modify and delete identity attributes and assign and unassign services. `IdRepo` is an abstract class that contains the methods that need to be implemented by plug-ins when building new adapters for repositories not currently supported. The current implementation supports Sun Java System Directory Server, IBM Tivoli Directory and Microsoft Active Directory. XXXXXX illustrates the design of the Identity Repository Service.

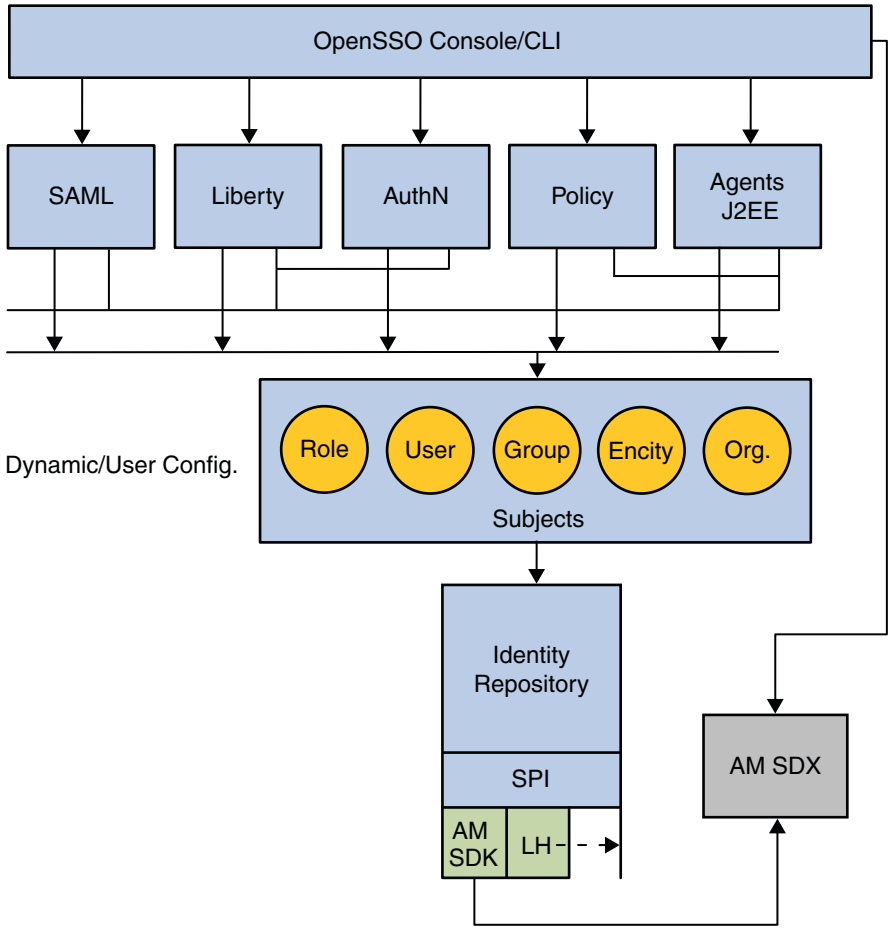


FIGURE 2–9 Identity Repository Service Design

Note – Administrator roles are also defined by the Identity Repository Service. (This is currently available only when the Sun Directory Server With FAM Core Services schema is loaded.) For example, the Realm Administrator can access all data in all configured realms while the Subrealm Administrator can access data only within the specified realm. For more information, see XXXXXXX Deployment Planning Guide. For information on realm privileges, see "Privileges" in *Sun Federated Access Manager 8.0 Administration Guide*.

Federation Services

OpenSSO Enterprise provides an open and extensible framework for identity federation and associated web services that resolve the problems of identity-enabling web services, web service discovery and invocation, security, and privacy. Federation Services are built on the following standards:

- Liberty Alliance Project Identity Federation Framework (Liberty ID-FF) 1.1 and 1.2
- OASIS Security Assertion Markup Language (SAML) 1.0 and 1.1
- OASIS Security Assertion Markup Language (SAML) 2.0
- WS-Federation (Passive Requestor Profile)

Federation Services allows organizations to share identity information (for example, which organizations and users are trusted, and what types of credentials are accepted) securely. Once this is enabled securely, federating identities is possible — allowing a user to consolidate the many local identities configured among multiple service providers. With one federated identity, the user can log in at one identity provider's site and move to an affiliated site without having to re-establish identity. Figure 2–10 illustrates the interactions between local and remote components of the Federation Services in a OpenSSO Enterprise deployment.

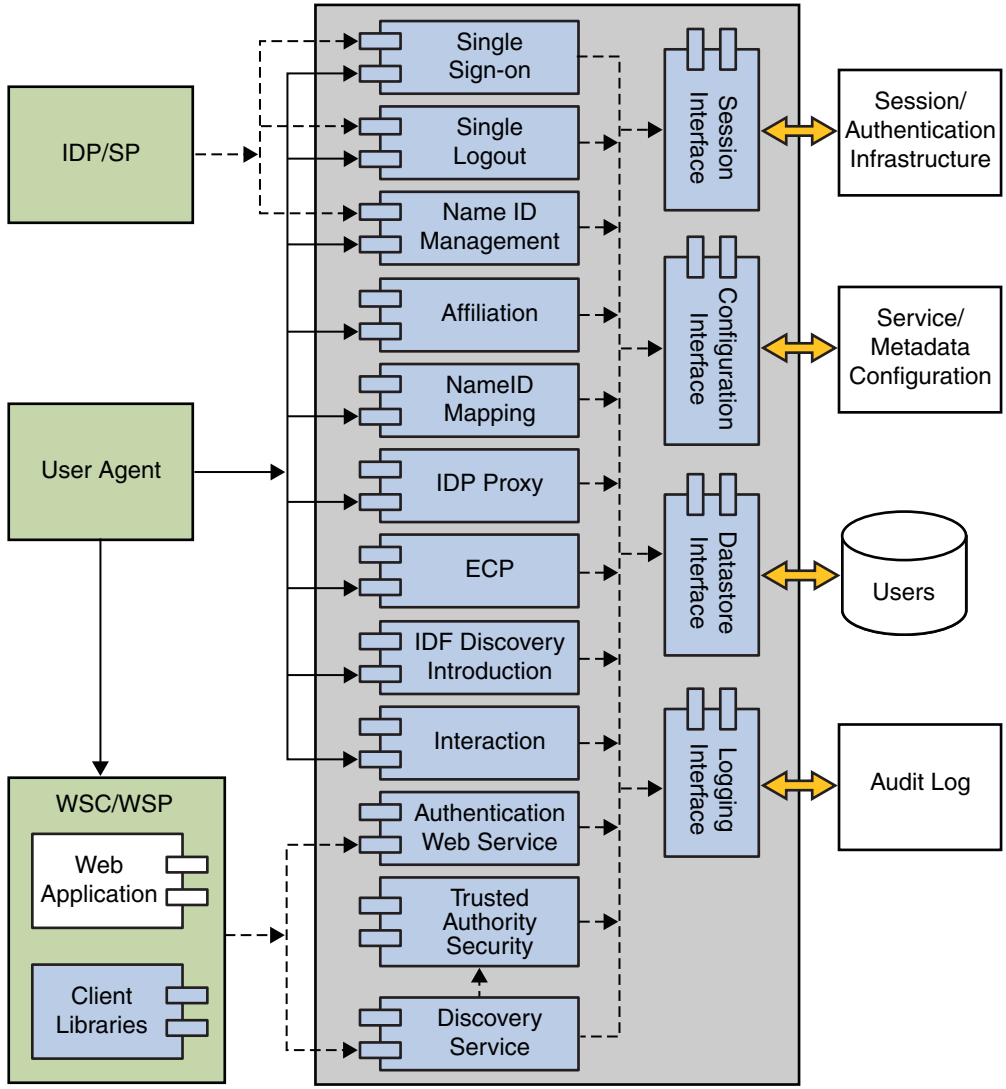


FIGURE 2–10 Federation Services Components within a OpenSSO Enterprise Deployment

More information on the Federation Services can be found in the Open Federation Architecture and the Federation Use Case documentation on the OpenSSO web site. Also, see Part III in this book.

Web Services Stack

The OpenSSO Enterprise Web Services Stack follows a standardized way of integrating web-based applications using XML, SOAP, and other open standards over an Internet Protocol (IP) backbone. They enable applications from various sources to communicate with each other because they are not tied to any one operating system or programming language. Businesses use web services to communicate with each other and their respective clients without having to know detailed aspects of each other's IT systems. OpenSSO Enterprise provides web services that primarily use XML and SOAP over HTTP. These web services are designed to be centrally provided in an enterprise's network for convenient access by client applications. OpenSSO Enterprise implements the follow web service specifications.

- Liberty Alliance Project Identity Web Services Framework (Liberty ID-WSF) 1.0, 1.1, and 2.0
- Web Services-Interoperability (WS-I) Basic Security Profile

The following table lists the OpenSSO Enterprise web services.

TABLE 2–1 OpenSSO Enterprise Web Services Stack

Web Service Name	Description
Authentication Web Service	Provides authentication to a web service client (WSC), allowing the WSC to obtain security tokens for further interactions with other services at the same provider. Upon successful authentication, the final Simple Authentication and Security Layer (SASL) response contains the resource offering for the Discovery Service.
Discovery Service	A web service that allows a requesting entity, such as a service provider, to dynamically determine a principal's registered attribute provider. Typically, a service provider queries the Discovery Service, which responds by providing a resource offering that describes the requested attribute provider. The implementation of the Discovery Service includes Java and web-based interfaces.
Liberty Personal Profile Service	A data service that supports storing and modifying a principal's identity attributes. Identity attributes might include information such as first name, last name, home address, and emergency contact information. The Liberty Personal Profile Service is queried or updated by a WSC acting on behalf of the principal.

TABLE 2–1 OpenSSO Enterprise Web Services Stack *(Continued)*

Web Service Name	Description
Security Token Service	The centralized Security Token Service that issues, renews, cancels, and validates security tokens.
SOAP Binding Service	A set of Java APIs implemented by the developer of a Liberty-enabled identity service. The APIs are used to send and receive identity-based messages using SOAP, an XML-based messaging protocol.

OpenSSO Enterprise uses both XML files and Java interfaces to manage web services and web services configuration data. A OpenSSO Enterprise XML file is based on the structure defined in the OpenSSO Enterprise Document Type Definition (DTD) files located in *path-to-context-root*`/fam/WEB-INF`. The main `sms.dtd` file defines the structure for all OpenSSO Enterprise service files (located in *path-to-context-root*`/fam/WEB-INF/classes`).

Caution – Do not modify any of the DTD files. The OpenSSO Enterprise API and their internal parsing functions are based on the default definitions and alterations to them may hinder the operation of the application.

For more information, see Part IV.

Web Services Security

In message security, security information is applied at the message layer and travels along with the web services message. Message layer security differs from transport layer security in that it can be used to decouple message protection from message transport so that messages remain protected after transmission, regardless of how many hops they travel on. This message security is available as Web Services Security in OpenSSO Enterprise and through the installation of an *authentication agent*. Web Services Security is the implementation of the WS-Security specifications and the Liberty Alliance Project Identity Web Services Framework (Liberty ID-WSF). Web Services Security allows communication with the Security Token Service to insert security tokens in outgoing messages and evaluate incoming messages for the same. Towards this end, authentication agents based on the Java Specification Request (JSR) 196 must be downloaded and installed on the web services client (WSC) machine and the web services provider (WSP) machine.

Note – JSR 196 agents can be used only on Sun Java System Application Server or Glassfish web containers.

To secure web services communications, the requesting party must first be authenticated with a security token which is added to the SOAP header of the request. Additionally, the WSC needs

to be configured to supply message level security in their SOAP requests and the WSP needs to be configured to enable message level security in their SOAP responses. Figure 2–11 illustrates the components used during a secure web services interaction.

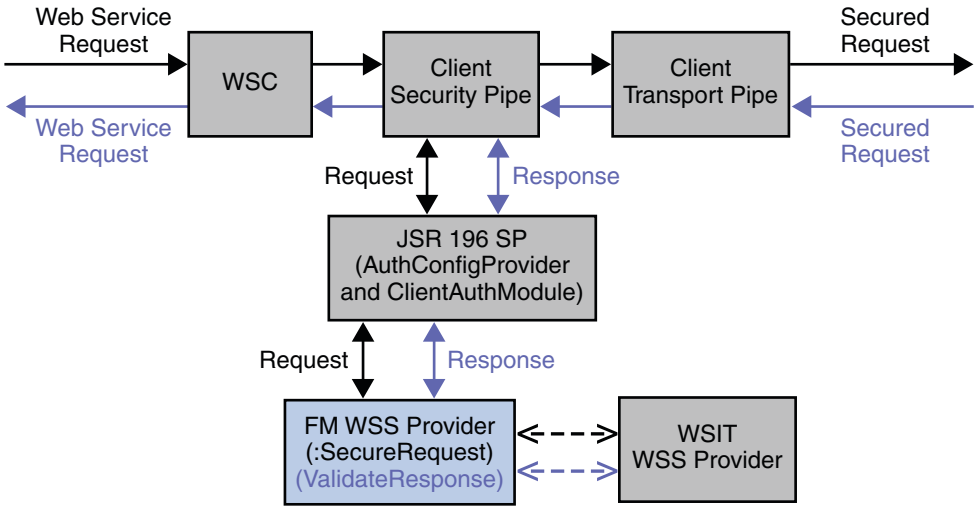


FIGURE 2–11 Web Services Security Components within a OpenSSO Enterprise Deployment

Note – The stand alone applications can directly invoke the interfaces (secure request by WSC, and validate response by WSP) from the WS-Security Library and establish message-level end-to-end web service security. Standalone Java applications do not need the WS-Security Provider Plugin.

For more information, see Part IV.

Identity Web Services

OpenSSO Enterprise contains client interfaces for authentication, authorization, session management, and logging in Java, C, and C++ (using a proprietary XML and SOAP over HTTP or HTTPs communication). These interfaces are used by policy agents and custom applications. Development using these interfaces, though, is labor-intensive. Additionally, the interfaces cause dependencies on OpenSSO Enterprise. Therefore, OpenSSO Enterprise has now implemented simple interfaces that can be used for:

- Authentication (verification of user credentials, password management)
- Authorization (policy evaluation for access to secured resources)

- Provisioning (self-registration, creating or deleting identity profiles, retrieve or update identity profile attributes)
- Token validation
- Search (return a list of identity profile attributes that match a search filter)

Note – Identity Web Services also interact with the Logging Service to audit and record Identity Web Services interactions.

These Identity Services are offered using either SOAP and the Web Services Description Language (WSDL) or Representational State Transfer (REST). With SOAP Identity Web Services, you point an integrated development environment (IDE) application project to the appropriate URL and generate the stub code that wraps the function calls to the services. (You can also use `wscompile`.) With REST Identity Web Services, no coding is necessary. It works right out of box.

Note – OpenSSO Enterprise supports Eclipse, NetBeans, and Visual Studio®.

When Identity Web Services have been implemented, a user interacts with the application which calls the identity repository to retrieve user profile data for authentication and personalization, the configuration data store to retrieve policy data for authorization, and the audit repository for log requests. The application authenticates, authorizes, audits, and finally creates personalized services for the user by calling either the SOAP/WSDL or REST Identity Web Service as provided by OpenSSO Enterprise.

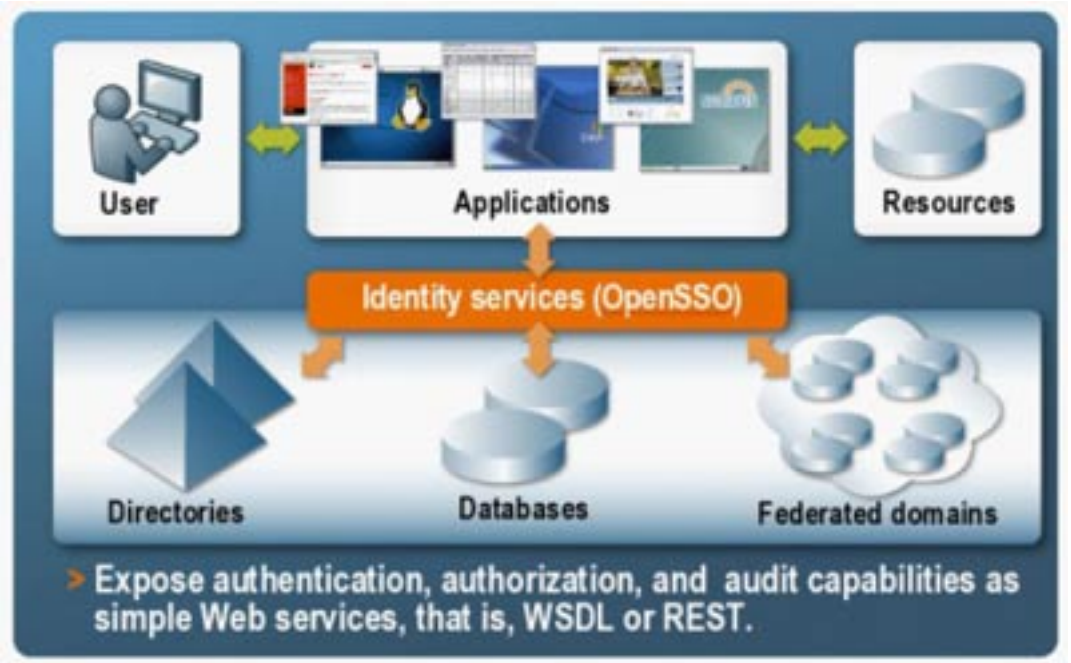


FIGURE 2–12 Basic Identity Web Services Process

For more information, see Part IV.

Global Services

Global services take configuration values and perform functions for OpenSSO Enterprise on a global basis. The following table lists the global services with brief descriptions.

TABLE 2–2 Global OpenSSO Enterprise Services

Service	What it Does
Common Federation Configuration	Contains configuration attributes for Federation Services.
Liberty ID-FF Service Configuration	Contains configuration attributes for the Liberty Alliance Project Identity Federation Framework.
Liberty ID-WSF Security Service	Contains configuration attributes for the Liberty Alliance Project Identity Web Services Framework.

TABLE 2–2 Global OpenSSO Enterprise Services *(Continued)*

Service	What it Does
Liberty Interaction Service	Contains configuration attributes for the Liberty Alliance Project Interaction Service — used to get consent from an owner to expose data, or to get additional data.
Multi-Federation Protocol	Contains configuration attributes for multi-federation protocol circles of trust.
Password Reset	Contains configuration attributes for the Password Reset Service.
Policy Configuration	Contains configuration attributes for the Policy Service.
SAML v2 Service Configuration	Contains configuration attributes for the SAML v2 interactions.
SAML v2 SOAP Binding	Contains configuration attributes for SAML v2 SOAP Binding Service.
Security Token Service	Contains configuration attributes for the Security Token Service.
Session	Contains configuration attributes for the Session Service.
User	Contains configuration attributes for user profiles.

Realms

A *realm* is the unit that OpenSSO Enterprise uses to organize configuration information. Authentication properties, authorization policies, data stores, subjects (including a user, a group of users, or a collection of protected resources) and other data can be defined within the realm. The data stored in a realm can include, but is not limited to:

- One or more subjects (a user, a group of users, or a collection of protected resources)
- A definition of one or more data stores to store subject (user) data
- Authentication details identifying, for example, the location of the authentication repository, and the type of authentication required.
- Policy information that will be used to determine which resources protected by OpenSSO Enterprise the subjects can access.
- Responder configurations that allows applications to personalize the user experience, once the user has successfully authenticated and been given access.
- Administration data for realm management

You create a top-level realm when you deploy OpenSSO Enterprise. The top-level realm (by default `opensso`) is the root of the OpenSSO Enterprise instance and contains OpenSSO Enterprise configuration data; it cannot be changed after it is created. In general, you should use the default root realm to configure identity data stores, and manage policies and authentication chains. During deployment, OpenSSO Enterprise creates a Realm Administrator who can perform all operations in the configured root realm, and a Policy Administrator who can only create and manage policies.

All other realms are configured under the `opensso` realm. These sub-realms may contain other sub-realms and so on. Sub-realms identify sets of users and groups that have different authentication or authorization requirements. The use of sub-realms should be restricted to the following two scenarios.

1. **Application Policy Delegation** The use case for this is when you need to have different Policy Administrators to create policies for a sub-set of resources. For example, let's assume a sub-realm is created and named Paycheck. This sub-realm is configured with a policy referral from the root realm for configuring protection of resources starting with https://paycheck.sun.com/paycheck. Within the Paycheck sub-realm, a Paycheck Administrator role or group is created and assigned Policy Administration privileges. These administrators are now able to login to the sub-realm and create policies for their applications. By default, the sub-realm inherits the same configuration data store and authentication chains configured for its parent; if these configurations change in the parent, a corresponding change would be needed in the sub-realm. Additionally, all users will still log in to the root realm for access to all the applications. The sub-realm is primarily for the Policy Administrator to manage policies for the application. An educated guess on the number of sub-realms that can be supported would be about 100.
2. **ISP/ASP/Silo** The use case for this scenario is when each sub-realm is to have its own set of identity data stores, authentication chains, and policies. Ideally the only common thread between the root and the sub-realm would be the referral policy created in the root realm to delegate a set of resources to the sub-realm. Users would not be able to log in to the root realm (unless they are a member) but would have to authenticate to their sub-realm. Also, agents would have to be configured to redirect user authentication to the particular sub-realm. With regards to performance, the most resource consuming component would be when persistent searches created by the data stores connect to the same directory. An educated guess on the number of sub-realms that can be supported would be about 50.

The OpenSSO Enterprise framework aggregates realm properties as part of the configuration data. Figure 2–13 illustrates how configuration data can use a hierarchy of realms to distribute administration responsibilities. Region 1, Region 2, and Region 3 are realms; Development, Operations, and Sales are realms sub to Region 3.

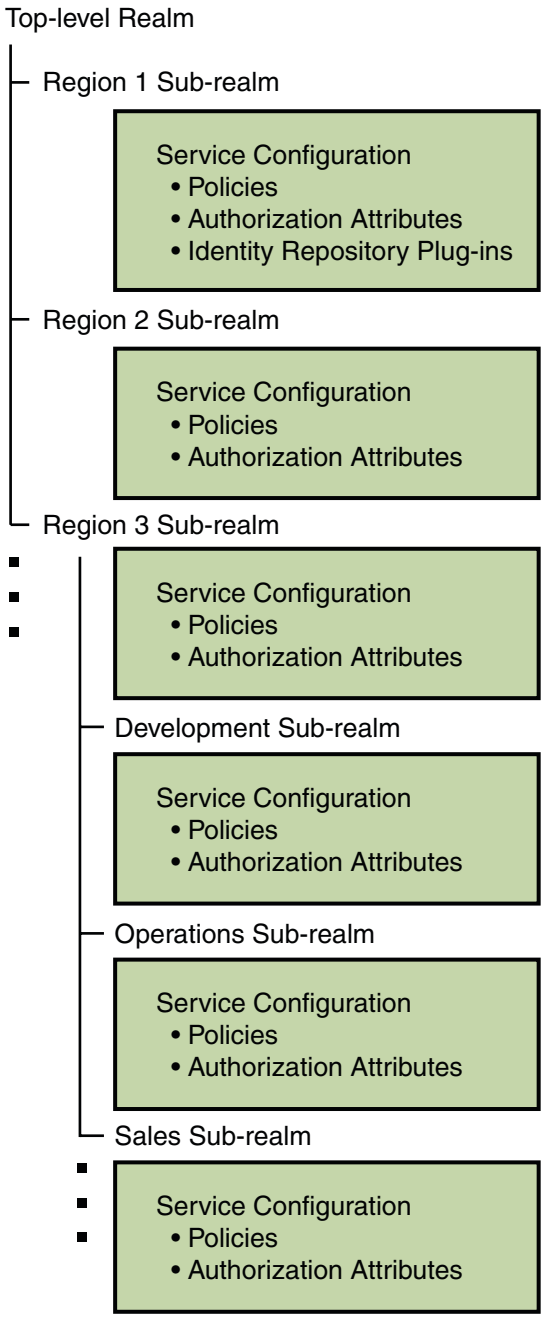


FIGURE 2–13 Realm Hierarchy for Configuration Data

Note – OpenSSO Enterprise 8.0 supports the Sun Java System Access Manager Legacy mode (which contains no realms) with a provided interface.

Additional Components

The following sections provide information on additional components used in a OpenSSO Enterprise deployment.

Data and Data Stores

OpenSSO Enterprise services need to interact with a number of different data stores. The following distinct repositories can be configured.

- A configuration repository provides server and service specific data.
- One or more identity repositories provide user profile information.
- Authentication repositories provide authentication credentials to a particular module of the Authentication Service.

A common LDAP connection pooling facility allows efficient use of network resources. In the simplest *demonstration* environment, a single LDAP repository is sufficient for all data however, the typical *production* environment tends to separate configuration data from other data. The following sections contain more specific information.

Configuration Data

The default configuration of OpenSSO Enterprise creates a branch in a fresh installation of a configuration data store for storing service configuration data and other information pertinent to the server's operation. OpenSSO Enterprise components and plug-ins access the configuration data and use it for various purposes including:

- Accessing policy data for policy evaluation.

- Finding location information for identity data stores and OpenSSO Enterprise services.
- Retrieving authentication configuration information that define how users and groups authenticate.
- Finding which partner servers can send trusted SAML assertions.

OpenSSO Enterprise supports Sun Java System Directory Server and the open source OpenDS as configuration data stores. Flat files (supported in previous versions of the product) are no longer supported but configuration data store failover is — using replication. Figure 2–14 illustrates how configuration data in the embedded data store is accessed.

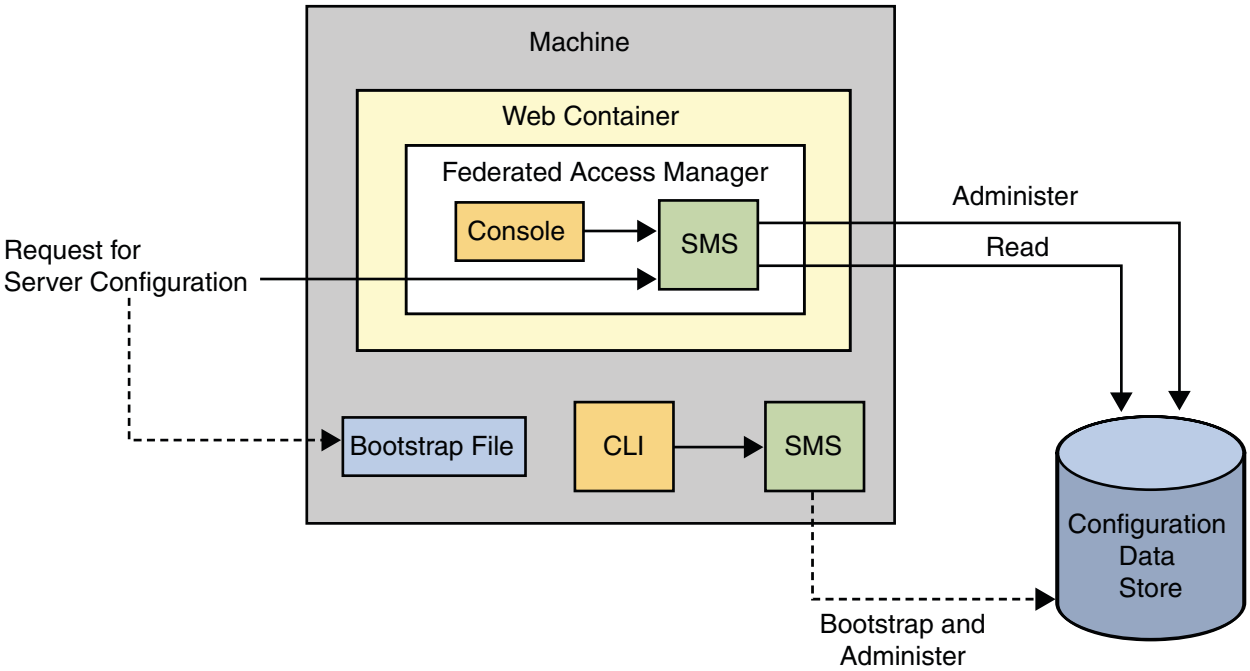


FIGURE 2–14 Accessing Configuration Data

Previous releases of Access Manager and Federation Manager stored product configuration data in a property file named `AMConfig.properties` that was installed local to the product instance directory. This file is deprecated for OpenSSO Enterprise on the server side although still supported for agents on the client side. See the *Sun OpenSSO Enterprise 8.0 Installation and Configuration Guide* for more information.

Configuration data comprises the attributes and values in the OpenSSO Enterprise configuration services, as well as default OpenSSO Enterprise users like `amadmin` and

anonymous. Following is a partial listing of the XML service files that contribute to the data. They can be found in the *path-to-context-root*/fam/WEB-INF/classes directory.

Note – The data in this node branch is private and is mentioned here for information purposes only.

- AgentService.xml
- amAdminConsole.xml
- amAgent70.xml
- amAuth.xml
- amAuth-NT.xml
- amAuthAD.xml
- amAuthAnonymous.xml
- amAuthCert.xml
- amAuthConfig.xml
- amAuthDataStore.xml
- amAuthHTTPBasic.xml
- amAuthJDBC.xml
- amAuthLDAP.xml
- amAuthMSISDN.xml
- amAuthMembership.xml
- amAuthNT.xml
- amAuthRADIUS.xml
- amAuthSafeWord-NT.xml
- amAuthSafeWord.xml
- amAuthSecurID.xml
- amAuthWindowsDesktopSSO.xml
- amClientData.xml
- amClientDetection.xml
- amConsoleConfig.xml
- amDelegation.xml
- amEntrySpecific.xml
- amFilteredRole.xml
- amG11NSettings.xml
- amLogging.xml
- amNaming.xml
- amPasswordReset.xml
- amPlatform.xml
- amPolicy.xml
- amPolicyConfig.xml
- amRealmService.xml
- amSession.xml
- amUser.xml
- amWebAgent.xml

- `idRepoEmbeddedOpenDS.xml`
- `idRepoService.xml`
- `identityLocaleService.xml`
- `ums.xml`

Caution – By default, the OpenSSO Enterprise configuration data is created and maintained in the embedded configuration data store apart from any identity data. Although users can be created in the configuration data store this is only recommended for demonstrations and development environments.

For more information, see "Embedded Configuration Data" on page 67.

Identity Data

An *identity repository* is a data store where information about users and groups in an organization is stored. User profiles can contain data such as a first name, a last name, a phone number, group membership, and an e-mail address; an identity profile template is provided out-of-the-box but it can be modified to suit specific deployments.

Identity data stores are defined per realm. Because more than one identity data store can be configured per realm OpenSSO Enterprise can access the many profiles of one identity across multiple data repositories. Sun Java System Directory Server with OpenSSO Enterprise Schema, Microsoft Active Directory, IBM Tivoli Directory and the AMSDK data store are the currently supported identity repositories. Plug-ins can be developed to integrate other types of repositories (for example, a relational database). Figure 2–15 illustrates a OpenSSO Enterprise deployment where the identity data and the embedded configuration data are kept in separate data stores.

verify support officially on Tivoli (Replaces generic LDAP v3), AD, Sun DS with FAM schema and amsdk

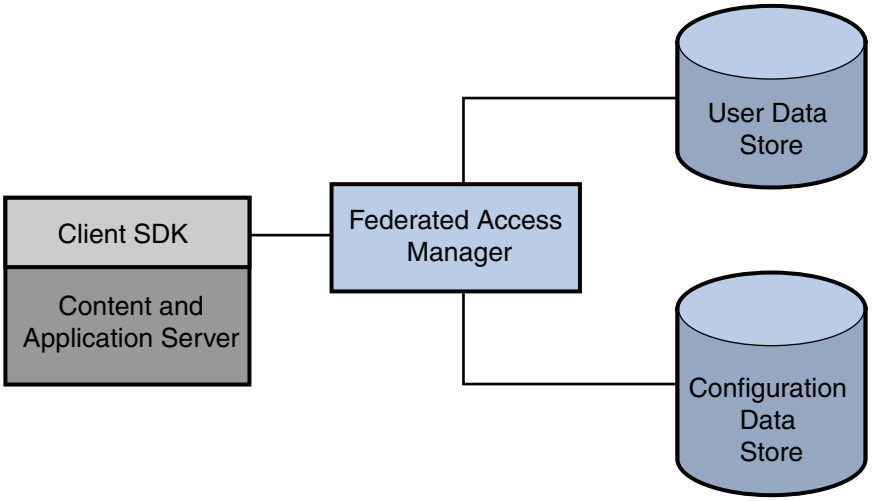


FIGURE 2–15 OpenSSO Enterprise Deployment with Two Data Stores

Note – The information in an identity repository is maintained by provisioning products separate from OpenSSO Enterprise. The supported provisioning product is Sun Java System Identity Manager.

OpenSSO Enterprise provides out-of-the-box plug-in support for some identity repositories. Each default plug-in configuration includes details about what operations are supported on the underlying data store. Once a realm is configured to use a plug-in, the framework can instantiate it and execute the operations on the appropriate identity repository. Each new plug-in developed must have a corresponding service management schema defining its configuration attributes. This schema would be integrated as a sub schema into `idRepoService.xml`, the service management file for the Identity Repository Service that controls the identity data stores available under a realm's Data Stores tab. The following sections contain information on the out-of-the-box plug-ins.

Remark 2–1 Reviewer **Please confirm these. I have gotten different comments from different reviewers.**

- "Generic Lightweight Directory Access Protocol (LDAP) version 3" on page 59
- "LDAPv3 Plug-in for Active Directory" on page 59
- "LDAPv3 Plug-in for Tivoli Directory" on page 59
- "Sun Directory Server With FAM Core Services" on page 59
- "Sun Directory Server With Full Schema (including Legacy)" on page 59
- "Access Manager Repository Plug-in" on page 59

Generic Lightweight Directory Access Protocol (LDAP) version 3

The Generic LDAPv3 identity repository plug-in can reside on an instance of any directory that complies with the LDAPv3 specifications. The underlying directory cannot make use of features that are not part of the LDAP version 3 specification, and no specific DIT structure can be assumed as LDAPv3 identity repositories are simply DIT branches that contain user and group entries. Each data store has a name that is unique among a realm's data store names, but not necessarily unique across all realms. The `com.sun.identity.idm.plugins.ldapv3.LDAPv3Repo` class provides the default LDAPv3 identity repository implementation. There are also implementations for Active Directory and IBM Tivoli Directory

LDAPv3 Plug-in for Active Directory

The Generic LDAPv3 identity repository plug-in was used to develop a default plug-in to write identity data to an instance of Microsoft® Active Directory®. The administration console provides a way to configure the directory but the schema needs to be loaded manually.

LDAPv3 Plug-in for Tivoli Directory

The Generic LDAPv3 identity repository plug-in was used to develop a default plug-in to write identity data to an instance of IBM Tivoli Directory®. The administration console provides a way to configure the directory but the schema needs to be loaded manually. XXXXXX Where is this config doc'ed?XXXXX

Sun Directory Server With FAM Core Services

This repository resides in an instance of Sun Java System Directory Server and holds the identity data. This option is available during the initial configuration of OpenSSO Enterprise.

Sun Directory Server With Full Schema (including Legacy)

This repository resides in an instance of Sun Java System Directory Server and holds the configuration data when installing OpenSSO Enterprise in Legacy and Realm mode. This option must be manually configured.

Access Manager Repository Plug-in

The Access Manager Repository can reside only in Sun Java System Directory Server and is used with the **Sun Directory Server With Access Manager Schema**. During installation, the repository is created in the same instance of Sun Java System Directory Server that holds the configuration data. The Access Manager Repository Plug-in is designed to work with Sun Java System Directory Server as it makes use of features specific to the server including *roles* and *class of service*. It uses a DIT structure similar to that of previous versions of Access Manager.

Note – This is no longer provided out of the box and many pieces are marked for deprecation. The Access Manager Repository is compatible with previous versions of Access Manager.

When you configure an instance of Access Manager in realm mode for the first time, the following occurs:

- An Access Manager Repository is created under the top-level realm.
- The Access Manager Repository is populated with internal Access Manager users.

Note – The Java Enterprise System installer does not set up an Access Manager Repository when you configure an Access Manager instance in legacy mode. Legacy mode requires an identity repository that is mixed with the Access Manager information tree under a single directory suffix.

Authentication Data

Authentication data contains authentication credentials for OpenSSO Enterprise users. An authentication data store is aligned with a particular authentication module, and might include:

- RADIUS servers
- SafeWord authentication servers
- RSA ACE/Server systems (supports SecurID authentication)
- LDAP directory servers

Note – Identity data may include authentication credentials although authentication data is generally stored in a separate authentication repository.

The `bootstrap` File

OpenSSO Enterprise uses a file to bootstrap itself. Previously, `AMConfig.properties` held configuration information to bootstrap the server but now a file named `bootstrap` points to the configuration data store allowing the setup servlet to retrieve the bootstrapping data. After deploying the OpenSSO Enterprise WAR and running the configuration wizard, configuration data is written to the configuration data store by the service management API contained in the Java package, `com.sun.identity.sm`. The setup servlet creates `bootstrap` in the top-level configuration directory. The content in bootstrap can be either of the following:

- A directory local to OpenSSO Enterprise (for example, `/export/SUNWam`) indicating the server was configured with a previous release. The directory is where `AMConfig.properties` resides.
- An encoded URL that points to a directory service using the following format:

```
ldap://ds-host:ds-port/server-instance-name?pwd=encrypted-amadmin-password&
embeddedds=path-to-directory-service-installation&basedn=base-dn&
dsmgr=directory-admin&dspwd=encrypted-directory-admin-password
```

For example:

```
ldap://ds.samples.com:389/http://owen2.red.sun.com:8080/fam?
pwd=AQIC5wM2LY4Sfcxi1dVZEdtfwar2vhWNkmS8&embeddedds=/fam/opends&
basedn=dc=fam,dc=java,dc=net&dsmgr=cn=Directory Manager
&dspwd=AQIC5wM2LY4Sfcxi1dVZEdtfwar2vhWNkmS8
```

where

- `ds.samples.com:389` is the host name and port of the machine on which the directory is installed.
- `http://owen2.red.sun.com:8080/fam` is the instance name.
- `AQIC5wM2LY4Sfcxi1dVZEdtfwar2vhWNkmS8` is the encrypted password of the OpenSSO administrator.
- `/fam/opends` is the path to the directory installation.
- `dc=fam,dc=java,dc=net` is the base DN.
- `cn=Directory Manager` is the directory administrator.
- `AQIC5xM2LY4SfcximdVZEdtfwar4vhWNkmG7` is the encrypted password for the directory administrator.

If more than one URL is present in the file and OpenSSO Enterprise is unable to connect or authenticate to the data store at the first URL, the bootstrapping servlet will try the second (and so on). Additionally, the number sign [#] can be used to exclude a URL as in:

```
# ldap://ds.samples.com:389/http://owen2.red.sun.com:8080/fam?
pwd=AQIC5wM2LY4Sfcxi1dVZEdtfwar2vhWNkmS8&embeddedds=/fam/opends&
basedn=dc=fam,dc=java,dc=net&dsmgr=cn=Directory+Manager
&dspwd=AQIC5wM2LY4Sfcxi1dVZEdtfwar2vhWNkmS8
```

Policy Agents

Policy agents are an integral part of SSO and CDSSO sessions. They are programs that police the web container on which resources are hosted. All policy agents interact with the Authentication Service in two ways:

- To authenticate itself in order to establish trust. This authentication happens using the Client SDK.
- To authenticate users having no valid session for access to a protected resource. This authentication happens as a browser redirect from the Distributed Authentication User Interface.

When a user requests access to a protected resource such as a server or an application, the policy agent intercepts the request and redirects it to the OpenSSO Enterprise Authentication Service for authentication. Following this, the policy agent requests the authenticated user's assigned policy and evaluates it to allow or deny access. (A *policy* defines the rules that specify a user's access privileges to a protected resource.) OpenSSO Enterprise supports two types of policy agents:

- The *web agent* enforces URL-based policy for C applications.
- The *Java EE agent* enforces URL-based policy and Java-based policy for Java applications on Java EE containers.

Both types of agents are available for you to install as programs separate from OpenSSO Enterprise. Policy agents are basically clients written using the Client SDK and Identity Services.

Note – All HTTP requests are implicitly denied unless explicitly allowed by the presence of a valid session and a policy allowing access. If the resource is defined in the Not Enforced list for the policy agent, access is allowed even if there is no valid session.

For more information, see J2EE Agent Architecture and Web Agent and C-API Architecture on the OpenSSO web site. For an overview of the available policy agents and links to specific information on installation, see the *Sun Java System Federated Access Manager Policy Agent 3.0 User's Guide*.

Authentication Agents

Authentication agents plug into web containers to provide message level security for web services, and supports both Liberty Alliance Project token profiles as well as Web Services-Interoperability Basic Security Profiles (WS-I BSP). (A *profile* defines the HTTP exchanges required to transfer XML requests and responses between web service clients and providers.) Authentication agents use an instance of OpenSSO Enterprise for all authentication decisions. Web services requests and responses are passed to the appropriate authentication modules using standard Java representations based on the transmission protocol. An HTTP Authentication Agent or a SOAP Authentication Agent can be used. For more information, see "Web Services Security" on page 47.

OpenSSO Enterprise Tools

Contained within the OpenSSO Enterprise ZIP are `famAdminTools.zip` and `famSessionTools.zip`. The following sections have some information about these tools.

`famadm` Command Line Interface

`famadm`, the command line interface (CLI), provides a second option to administer OpenSSO Enterprise using the command line. For example, `famadm` can be used to create a policy or import and export Liberty ID-FF metadata. `famadm` is the recommended way for batch processing. It is in `famAdminTools.zip`. For more information, see Chapter 4, "Installing the OpenSSO Enterprise Utilities and Scripts," in *Sun OpenSSO Enterprise 8.0 Installation and Configuration Guide* and Part I, "Command Line Interface Reference," in *Sun Federated Access Manager Administration Reference*.

Session Failover Tools

`famSessionTools.zip` contains scripts and binaries for setting up session failover and databases. For more information, see Chapter 5, "Implementing OpenSSO Enterprise Session Failover," in *Sun OpenSSO Enterprise 8.0 Installation and Configuration Guide*.

Client SDK

Enterprise resources cannot be protected by OpenSSO Enterprise until the OpenSSO Enterprise Client SDK is installed on the machine that contains the resource that you want to protect. (The Client SDK is automatically installed with a policy agent.) The Client SDK allows you to customize an application by enabling communication with OpenSSO Enterprise for retrieving user, session, and policy data. For more information, see Chapter 1, "Enhancing Remote Applications Using the Client Software Development Kit," in *Sun Federated Access Manager 8.0 Developer's Guide* and the *Federated Access Manager 8.0 Java API Reference*.

Service Provider Interfaces for Plug-ins

The OpenSSO Enterprise service provider interfaces (SPI) can be implemented as plug-ins to provide customer data to the OpenSSO Enterprise framework for back-end processing. Some customer data comes from external data base applications such as identity repositories while other customer data comes from the OpenSSO Enterprise plug-ins themselves. You can develop additional custom plug-ins to work with the SPI. For a complete list of the SPI, see the *Federated Access Manager 8.0 Java API Reference*. Additional information can be found in the *Sun Federated Access Manager 8.0 Developer's Guide*. The following sections contain brief descriptions.

- "Authentication Service SPI" on page 64
- "Federation Service SPI" on page 64
- "Identity Repository Service SPI" on page 64
- "Policy Service SPI" on page 64
- "Service Configuration Plug-in" on page 64

Authentication Service SPI

The `com.sun.identity.authentication.spi` package provides interfaces and classes for writing a supplemental authentication module to plug into OpenSSO Enterprise. The `com.sun.identity.authentication` package provides interfaces and classes for writing a remote client application that can access user data in a specified identity repository to determine if a user's credentials are valid.

Federation Service SPI

The `com.sun.identity.federation.services` package provides plug-ins for customizing the Liberty ID-FF profiles implemented by OpenSSO Enterprise. The `com.sun.identity.federation.plugins` package provides an interface that can be implemented to perform user specific processing on the service provider side during the federation process. The `com.sun.identity.saml2.plugins` package provides the SAML v2 service provider interfaces (SPI). The `com.sun.identity.wsfederation.plugins` package provides the WS-Federation based SPI.

Identity Repository Service SPI

The `com.sun.identity.idm` package contains the `IdRepo` interface that defines the abstract methods which need to be implemented or modified by Identity Repository Service plug-ins. The `com.sun.identity.plugin.datastore` package contains interfaces that search for and return identity information such as user attributes and membership status for purposes of authentication.

Policy Service SPI

The `com.sun.identity.policy.interfaces` package provides interfaces for writing custom policy plug-ins for Conditions, Subjects, Referrals, Response Providers and Resources.

Service Configuration Plug-in

The `com.sun.identity.plugin.configuration` package provides interfaces to store and manage configuration data required by the core OpenSSO Enterprise components and other plug-ins.

Note – In previous releases, the functionality provided by the Service Configuration plug-in was known as the Service Management Service (SMS).

◆ ◆ ◆ CHAPTER 3

Simplifying OpenSSO Enterprise

This chapter contains information on the usability and manageability features of OpenSSO Enterprise. It includes the following sections:

Installation and Configuration

Previous versions of Sun Microsystems' access management server product were built for multiple hardware platforms, and different web containers. The complexity of this development process led to the release of separate platform and container patches. To alleviate this extraneous development, OpenSSO Enterprise is now available as a single ZIP file which can be downloaded, unzipped, and quickly deployed; there will be no separate installations for each hardware platform. The ZIP file will contain the full OpenSSO Enterprise web archive (WAR), layouts for the generation of other specific WARs, libraries, the Java API reference documentation, and samples. (Tools for use with OpenSSO Enterprise, including the command line interfaces and policy and authentication agents, can be downloaded separately.) Figure 3–1 illustrates the process for deployment, installation and configuration of a new OpenSSO Enterprise WAR and a patched WAR.

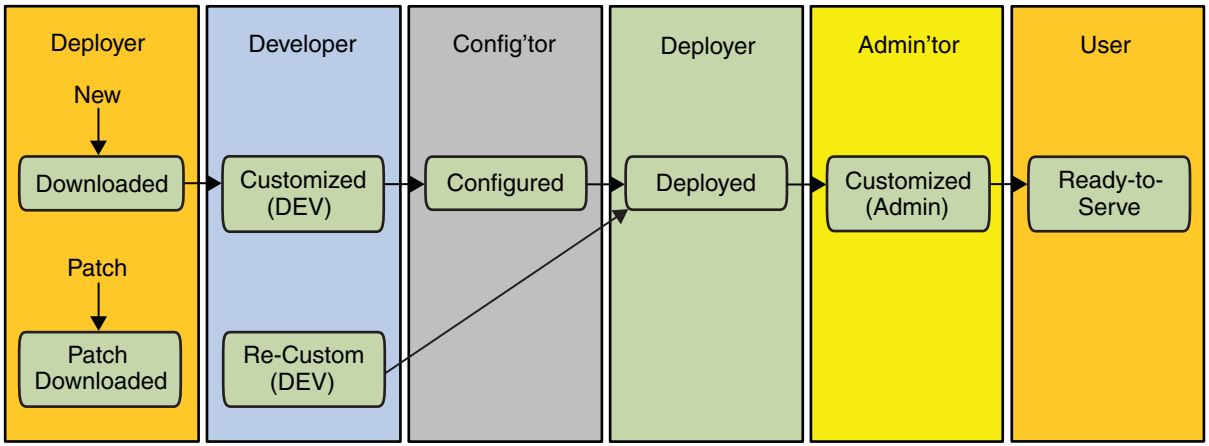


FIGURE 3–1 Customizing, Patching and Deploying the OpenSSO Enterprise WAR

Note – When patched, a full patched version of the OpenSSO Enterprise WAR will be included in the download, assuring that there is always a single download to get the latest bits.

The intent of this new process is to allow the administrator to download OpenSSO Enterprise and deploy it on the container or platform of choice, using the web container's administration console or command line interfaces. After the initial launch of the deployed WAR, the user is directed to a JavaServer Page (JSP) called the `Configurator` that prompts for configuration parameters (including, but not limited to, the host name, port number, URI, and repositories), provides data validation for the parameter values to prevent errors, and eliminates post-installation configuration tasks. Once successfully configured, any further changes to the configuration data store must be made using the OpenSSO Enterprise console or command line interfaces.

Note – When deploying OpenSSO Enterprise against an existing legacy installation, the Directory Management tab will be enabled in the new console.

For more information including a directory layout of the ZIP, see the *Sun OpenSSO Enterprise 8.0 Installation and Configuration Guide*.

Embedded Configuration Data

OpenSSO Enterprise has implemented an embedded configuration data store to replace the `AMConfig.properties` and `serverconfig.xml` files which had been the storage files for server configuration data. Previously, each instance of the server installed had separate configuration files but now when deploying more than one instance of OpenSSO Enterprise, all server configuration data is stored centrally, in one embedded configuration data store per instance. After the OpenSSO Enterprise WAR is configured, a sub configuration is added under the Platform Service to store the data and a bootstrap file that contains the location of the configuration data store is created in the installation directory. Figure 3–2 illustrates how OpenSSO Enterprise is bootstrapped.

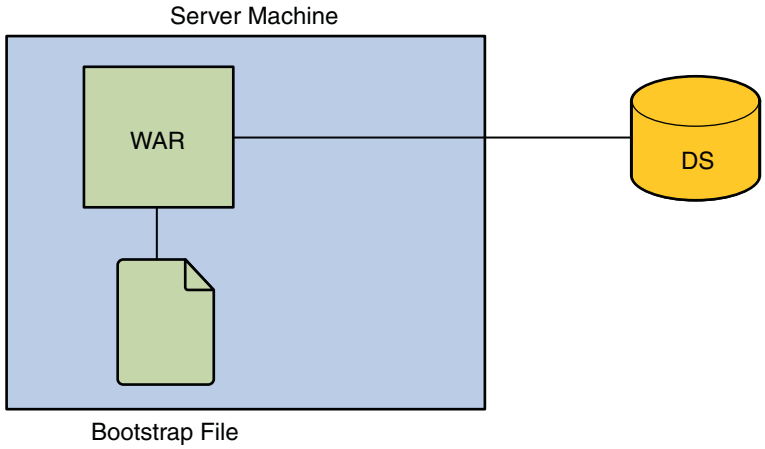


FIGURE 3–2 Bootstrapping OpenSSO Enterprise

Post-installation, the configuration data can be reviewed and edited using the administration console or the `famadm` command line interface. For more information see the *Sun OpenSSO Enterprise 8.0 Installation and Configuration Guide* and the *Sun Federated Access Manager 8.0 Administration Guide*.

Note – OpenSSO Enterprise also supports an LDAPv3–based solution that uses an existing directory server for configuration data storage. This is configured during installation. Supported directories include Sun Java System Directory Server, Microsoft Active Directory, and IBM Tivoli Directory.

Centralized Agent Configuration

Policy agents function based on a set of configuration properties. Previously, these properties were stored in the `AMAgent.properties` file, residing on the same machine as the agent. With Centralized Agent Configuration, OpenSSO Enterprise moves most of the agent configuration properties to the embedded configuration data store. Now agent profiles can be configured to store properties locally (on the machine to which the agent was deployed) or centrally (in the embedded configuration data store), making this new function compatible with both older 2.x agents and newer 3.0 agents. Following is an explanation of the local and central agent configuration repositories.

- Local agent configuration is supported for backward compatibility. Agent configuration data is stored in a property file named `FAMAgentConfiguration.properties` that is stored on the agent machine. It is only used by agent profiles configured locally.
- Centralized Agent Configuration stores agent configuration data in a centralized data store managed by OpenSSO Enterprise. When an agent starts up, it reads its bootstrapping file to initialize itself. `FAMAgentBootstrap.properties` is stored on the agent machine and indicates the location from where the configuration properties need to be retrieved. It is used by agent profiles configured locally or centrally. Based on the repository setting in `FAMAgentBootstrap.properties`, it retrieves the rest of its configuration properties. If the repository is local, it reads the agent configuration from a local file; if the repository is remote, it fetches its configuration from OpenSSO Enterprise.

Thus, Centralized Agent Configuration separates the agent configuration properties into two places: a bootstrapping file stored local to the agent and either a local (to the agent) or central (local to OpenSSO Enterprise) agent configuration data store.
`FAMAgentBootstrap.properties` is the bootstrapping file used by agent profiles configured locally or centrally. It is stored on the agent machine and indicates the local or central location from where the agent's configuration properties are retrieved. If the repository is local to the agent, it reads the configuration data from a local file; if the repository is remote, it fetches its configuration from OpenSSO Enterprise. Choosing Centralized Agent Configuration provides an agent administrator with the means to manage multiple agent configurations from a central place using either the OpenSSO Enterprise console or command line interface. Figure 3–3 illustrates how an agent retrieves bootstrapping and local configuration data, and configuration data from the configuration data store.

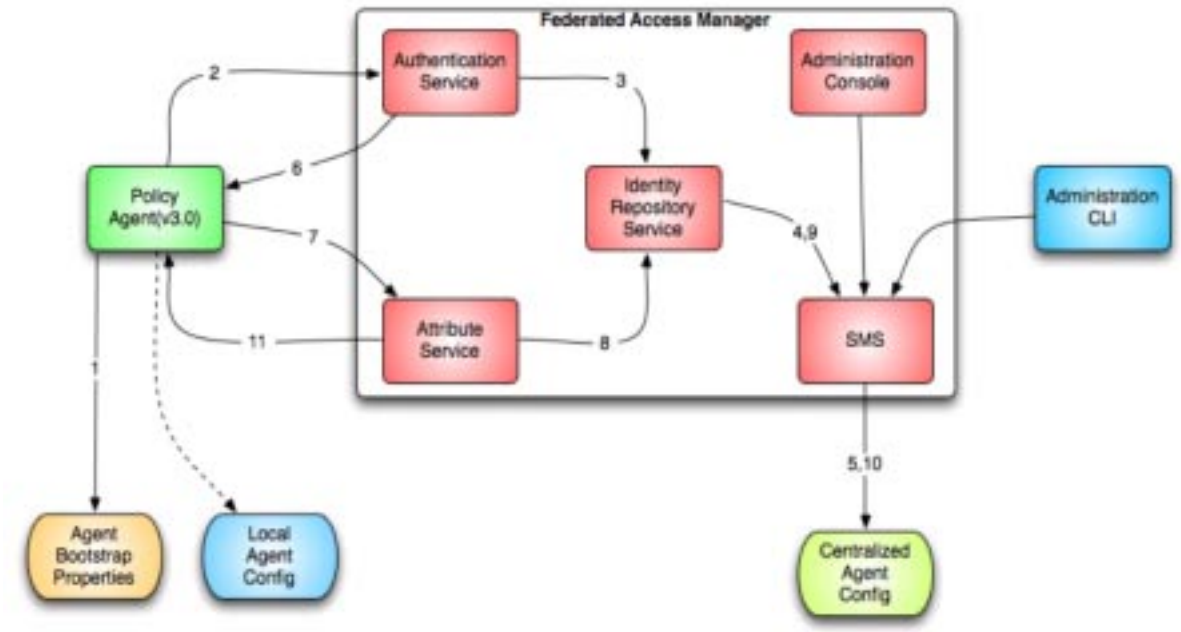


FIGURE 3–3 Retrieving Agent Configuration Data

An agent fetches its configuration properties periodically to determine if there have been any configuration changes. Any agent configuration changes made centrally are conveyed to the affected agents which will react accordingly based on the nature of the updated properties. If the properties affected are *hot swappable*, the agent can start using the new values without a restart of the underlying agent web container. Notification of the agent when configuration data changes and polling by the agent for configuration changes can be enabled. Agents can also receive notifications of session and policy changes.

Note – A agent configuration data change notification does not contain the actual data; it is just a ping that, when received, tells the agent to make a call to OpenSSO Enterprise and reload the latest. Session and policy notifications, on the other hand, contain the actual data changes. Also, when using a load balancer, the notification is sent directly to the agent whose configuration has been changed. It does not go through the load balancer.

For more information see the *Sun Federated Access Manager 8.0 Administration Guide*.

Common Tasks

OpenSSO Enterprise has implemented a Common Tasks tab that allows an administrators to create federation-based objects using console wizards. The wizards offer simplified provider configuration with metadata input using the URL `http://fam.sun.com:8080/fam/saml2/jsp/exportmetadata.jsp` or a metadata file. The following things can be done using a Common Task wizard:

- **Create SAML v2 Providers** They can be hosted or remote provider; and identity or service provider. To create them, you just need to provide some basic information about the providers.
- **Create Fedlet** A *Fedlet* is a small ZIP file that can be given to a service provider to allow for immediate federation with an identity provider configured with OpenSSO Enterprise. It is ideal for an identity provider that needs to enable a service provider with no federation solution in place. The service provider simply adds the Fedlet to their application, deploys their application, and they are federation enabled. For more information, see "The Fedlet" on page 148.
- **Test Federation Connectivity** This task validates your federation configuration. It will show if federation connections are being made successfully by identifying where the troubles, if any, are located.
- **Access Documentation** This link opens the OpenSSO documentation page. View frequently asked questions, tips, product documentation, engineering documentation as well as links to the community blogs.
- **Register Your Product** This link allows you to register your product with Sun Connection. You must have a Sun Online Account in order to complete the registration. If you do not already have one, you may request one as part of this process.

Figure 3–4 is a screen capture of the Common Tasks wizard.

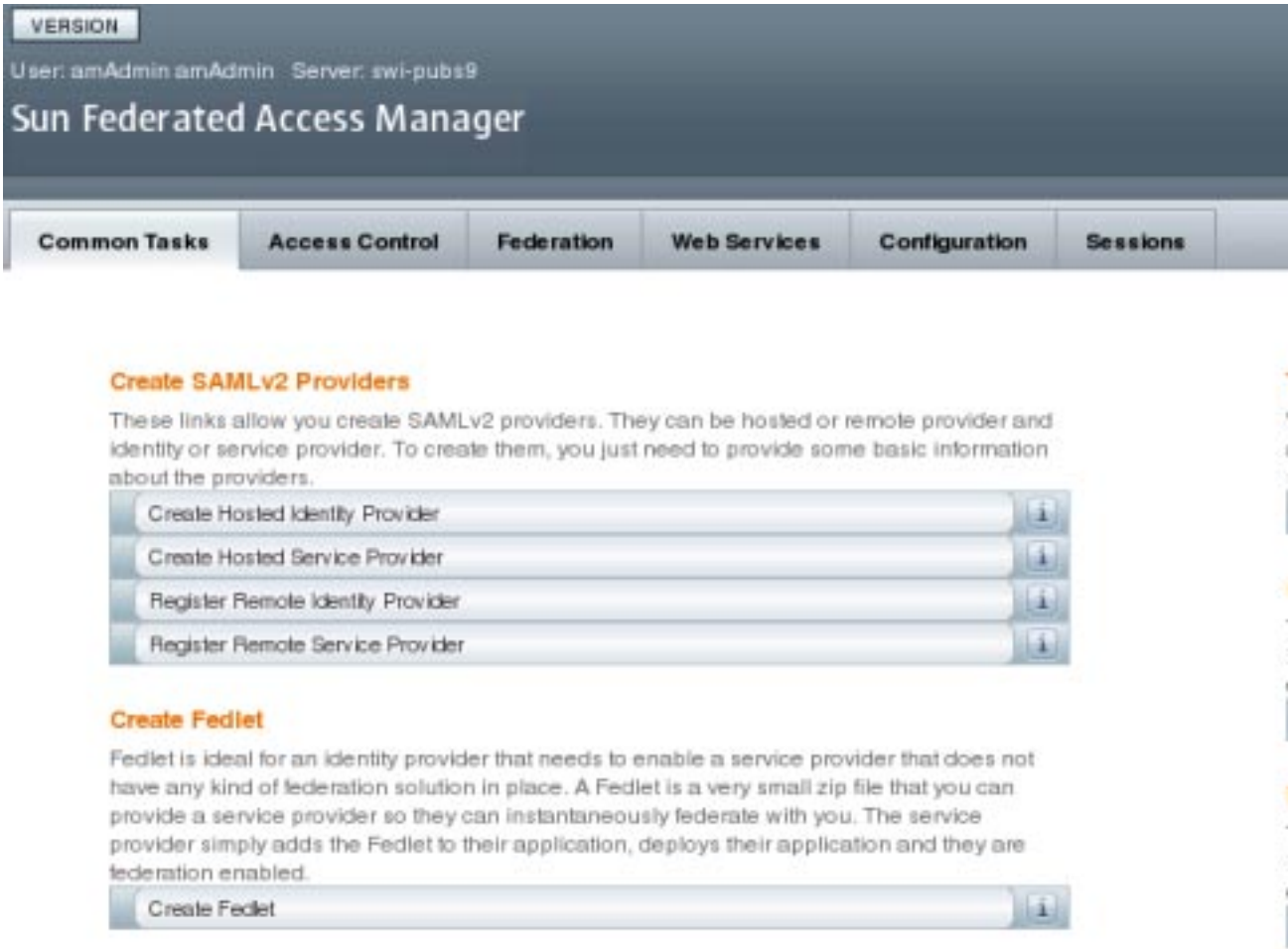


FIGURE 3–4 The Common Tasks Wizard

For more information see the *Sun Federated Access Manager 8.0 Administration Guide*.

Third Party Integration

OpenSSO Enterprise makes it easy to integrate with third-party software. Plug-ins and other tools have been developed to ease the integration of OpenSSO Enterprise and the following products.

- "Sun Java System Identity Manager" on page 72
- "Computer Associates SiteMinder" on page 72

- "Oracle Access Manager" on page 72

For more information, see *Sun Java System Federated Access Manager Integration Guide*.

Sun Java System Identity Manager

Sun Java System Identity Manager enables an enterprise to manage and audit access to accounts and resources as well as distribute the access management overhead. A OpenSSO Enterprise policy agent is deployed on the Identity Manager machine to regulate access to the Identity Manager server. By mapping Identity Manager objects to OpenSSO Enterprise users and resources, you may significantly increase operational efficiency. For use cases, a technical overview, installation and configuration procedures, architecture diagrams and process flows, see Chapter 1, "Integrating Sun Identity Manager ," in *Sun Java System Federated Access Manager Integration Guide*.

Computer Associates SiteMinder

Computer Associates SiteMinder (originally developed by Netegrity) is one of the industry's first SSO products — used in a majority of legacy web SSO deployments to protect their intranet and external applications. OpenSSO Enterprise provides the tools for SSO integration with SiteMinder in both intranet and federated environments. They include a SiteMinder Agent and a OpenSSO Enterprise Authentication Module for SiteMinder. They can be found in the `integrations/siteminder` directory of the exploded `fam.war`. For use cases, a technical overview, installation and configuration procedures, architecture diagrams and process flows, see Chapter 2, "Integrating CA SiteMinder," in *Sun Java System Federated Access Manager Integration Guide*.

Oracle Access Manager

Oracle Access Manager (originally developed by Oblix) is an SSO product with many of the same features as Sun OpenSSO Enterprise and Computer Associates SiteMinder. Oracle Access Manager can be deployed to protect both internal and external applications. OpenSSO Enterprise provides an Oracle Agent and a custom OpenSSO Enterprise Authentication Module for Oracle Access Manager. They can be found in the `integrations/oracle` directory of the exploded `fam.war`. For use cases, a technical overview, installation and configuration procedures, architecture diagrams and process flows, see Chapter 3, "Integrating Oracle Access Manager," in *Sun Java System Federated Access Manager Integration Guide*.

◆ ◆ ◆ CHAPTER 4

Deploying OpenSSO Enterprise

Sun OpenSSO Enterprise can be deployed in a number of ways. This chapter contains two sample deployment architectures.

For more information, see XXXXXXXX. Deployment Planning Guide.

Deployment Architecture 1

Figure 4–1 illustrates one deployment architecture for Sun OpenSSO Enterprise.

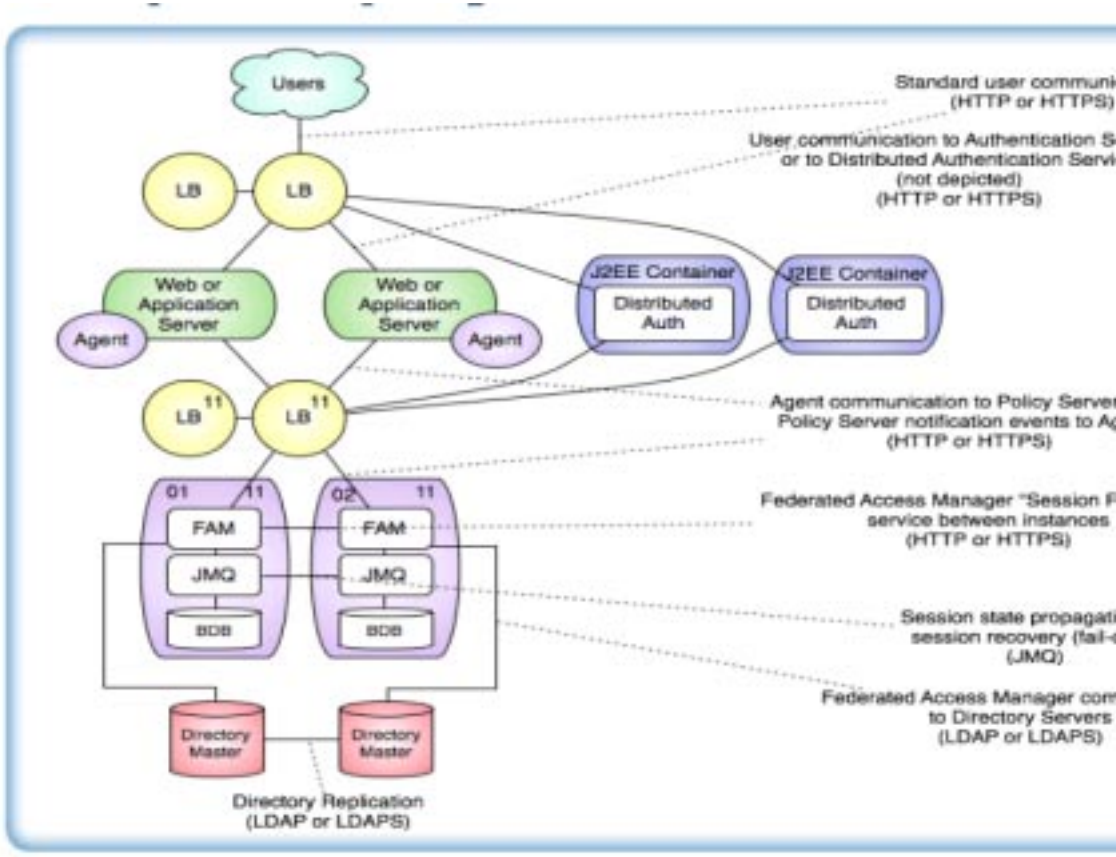


FIGURE 4–1 Sample Deployment Architecture 1

Deployment Architecture 2

Figure 4–2 illustrates another deployment architecture for Sun OpenSSO Enterprise.

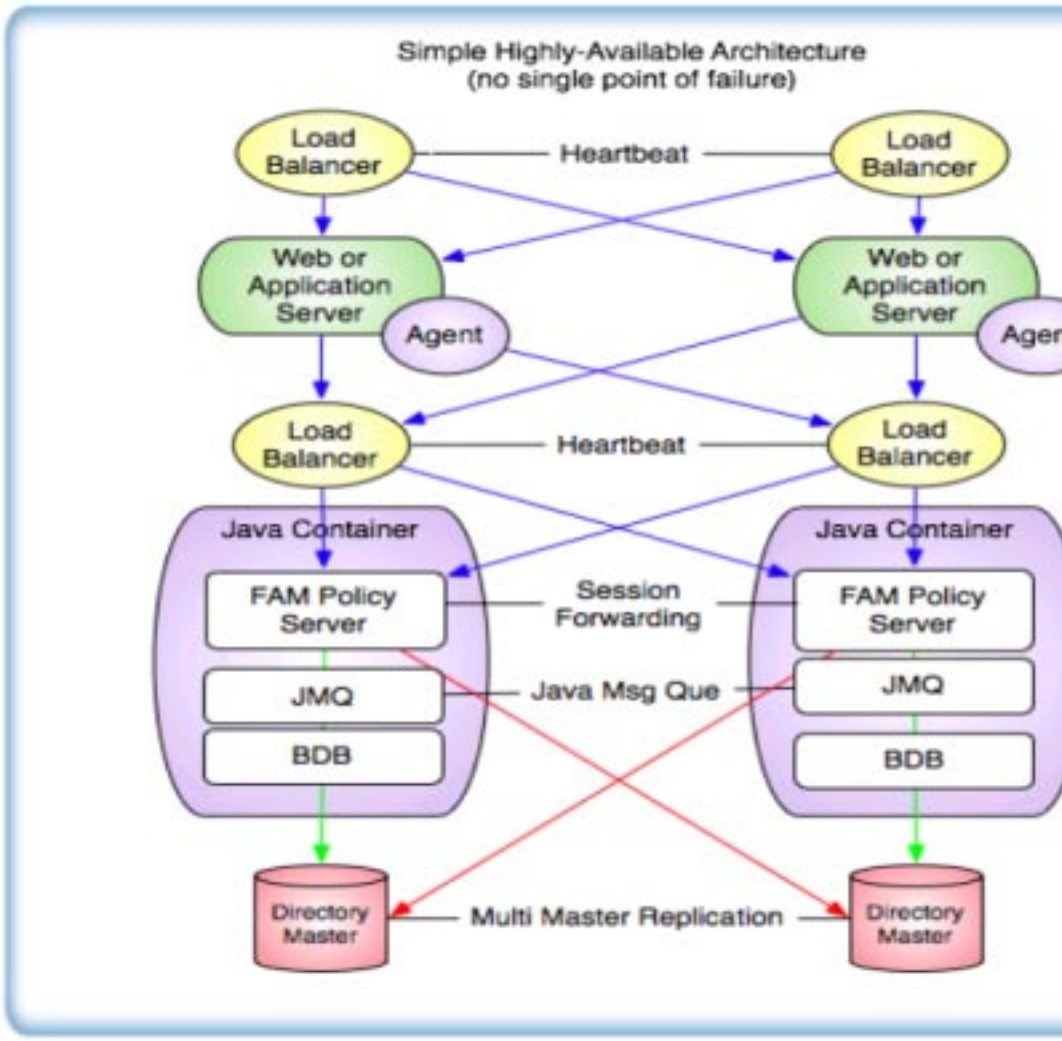


FIGURE 4–2 Sample Deployment Architecture 2

◆ ◆ ◆ CHAPTER 5

Recording Events with the Logging Service

Sun OpenSSO Enterprise provides its own logging feature that records information such as user login, user logout, session creation, and policy evaluation. This chapter describes how OpenSSO Enterprise logging works. It contains the following sections:

Logging Service Overview

The Logging Service enables OpenSSO Enterprise components to record information such as access denials and approvals, authentication events, and authorization violations. Administrators can use the logs to track user actions, analyze traffic patterns, audit system usage, review authorization violations, and troubleshoot. The logged information is recorded in one centralized directory. The Client SDK enables external applications to access the Logging Service. This section contains the following:

About the Logging Service

The purpose of the Logging Service is to provide the facilities to record events that can then be used to assign responsibility for actions occurring through OpenSSO Enterprise. For example, an individual's attempts to compromise the security of OpenSSO Enterprise, and to what extent those attempts penetrate, can be monitored. A global service configuration file named

amLogging.xml defines the Logging Service attributes. These attributes include configuration information such as maximum log size, log location, and log format (flat file or relational database). The attribute values are applied across the OpenSSO Enterprise deployment and inherited by every configured realm. The structure of amLogging.xml is defined by file sms.dtd.

Note – The Logging Service is fundamentally an extension of the java.util.logging.LogManager, java.util.logging.Logger, java.util.logging.LogRecord, java.util.logging.Formatter and java.util.logging.Handler classes.

Configuring the Logging Service

When OpenSSO Enterprise starts or when any logging configuration data is changed using the administration console, the Logging Service configuration data is loaded (or reloaded) into the Logging Service. This data includes the log message format, maximum log size, and the number of history files. Authenticated and authorized entities (for example, an application) can then use the Client SDK to access the Logging Service features from a local or remote server. The Client SDK uses an XML over HTTP layer to send logging requests to the Logging Service on the server where OpenSSO Enterprise is installed.

Recording Events

Log records are created using the com.sun.identity.log.LogRecord class, and then logged by authenticated and authorized entities using the com.sun.identity.log.Logger class. Log records can be logged by:

- Other components of the OpenSSO Enterprise server.
- Utilities installed on the OpenSSO Enterprise server system.
- Other OpenSSO Enterprise servers using a second instance of OpenSSO Enterprise acting as the *log server*.
- Remote client applications (for example, policy agents) using the OpenSSO Enterprise Logging Service.

The following table summarizes the default items logged in the LogRecord.

TABLE 5–1 Events Recorded in LogRecord

Event	Description
Time	The date (YYYY-MM-DD) and time (HH:MM:SS) at which the log message was recorded. This field is not configurable.

TABLE 5–1 Events Recorded in `LogRecord` *(Continued)*

Event	Description
Data	Variable data pertaining to the log records's `MESSAGE ID`. This field is not configurable.
ModuleName	Name of the OpenSSO Enterprise service or application being logged. Additional information on the value of this field can be found in "Adding Log Data" on page 88.
Domain	OpenSSO Enterprise domain to which the user (whom the log record is regarding) belongs. This information is taken from the session token passed in the `LogRecord(level,msg,token)` call.
LogLevel	The Java 2 Platform, Standard Edition (J2SE) version 1.4 log level of the log record.
LoginID	The identifier of the user (taken from the session token) as the subject of the log record.
IPAddress	IP address from which the operation was performed.
LoggedBy	User who writes the log record. The information is taken from the session token passed during `logger.log(logRecord, ssoToken)`.
HostName	Host name associated with the IP address above. This is present if the Log Record Resolve Host Name attribute is enabled. If not, the IP address is printed.
MESSAGEID	Non-internationalized message identifier for this log record's message.
ContextID	Session identifier associated with a particular login session. The session identifier is for the entity about whom the log record is regarding.

Log File Formats and Log File Types

The following sections contain information about OpenSSO Enterprise log files:

Log File Formats

Log records generated for one event are entered as two separate records. The first log record records the attempt to perform an action; the second log record records the result of the attempt. The following example illustrates this two record approach.

EXAMPLE 5–1 Log Record Example

```
Data:      agroupSubscription1|group|/
MessageID:     CONSOLE-1
```

EXAMPLE 5–1 Log Record Example *(Continued)*

and

```
Data:     agroupSubscription1|group|/
MessageID:    CONSOLE-2
```

In this example, `CONSOLE-1` indicates an attempt to create an identity object, and `CONSOLE-2` indicates that the attempt to create the identity object was successful. The root organization is noted by a forward slash (`/`). The variable parts of the messages (*agroupSubscription1*, *group*, and /) are separated by a pipe character (`|`) and continue to go into the `Data` field of each log record. The `MessageID` string is not internationalized in order to facilitate machine-readable analysis of the log records in any locale. OpenSSO Enterprise can record events in either of the following formats:

- "Flat File Format" on page 80
- "Relational Database Format" on page 80

Flat File Format

The default flat file format is the W3C Extended Log Format (ELF). OpenSSO Enterprise uses this format to record the default fields in each log record. See "Recording Events" on page 78 for a list of default fields and their descriptions. Example 5–2 illustrates an authentication log record formatted for a flat file. The fields are in this order: `Time`, `Data`, `ModuleName`, `MessageID`, `Domain`, `ContextID`, `LogLevel`, `LoginID`, `IPAddr`, `LoggedBy`, and `HostName`.

EXAMPLE 5–2 Flat File Record From `amAuthentication.access`

```
"2005-08-01 16:20:28"   "Login Success" LDAP    AUTHENTICATION-100
   dc=example,dc=com       e7aac4e717dda1bd01      INFO
uid=amAdmin,ou=People,dc=example,dc=com 192.18.187.152
"cn=exampleuser,ou=Example Users,dc=example,dc=com" exampleHost
```

Relational Database Format

When OpenSSO Enterprise uses a relational database to log messages, the messages are stored in a database table. OpenSSO Enterprise uses Java Database Connectivity (JDBC), which provides connectivity to a wide range of databases. (Oracle® and MySQL databases are currently supported.) Table 5–2 summarizes the schema for a relational database.

TABLE 5–2 Relational Database Log Format

Column Name	Data Type	Description
TIME	Date (Oracle) DateTime (MySQL)	The format is YYYY-MM-DD HH24:MI:SS (Oracle) or %Y-%m-%d %H:%i:%s (MySQL). The formats are specified in the Logging Service attributes.
DATA	CLOB (Oracle) LONGTEXT (MySQL)	The data type is specified in the Logging Service attributes.
MODULENAME	VARCHAR(255)	Name of the OpenSSO Enterprise component invoking the log record.
DOMAIN	VARCHAR(255)	OpenSSO Enterprise domain of the user.
LOGLEVEL	VARCHAR(255)	JDK 1.4 log level of the log record.
LOGINID	VARCHAR(255)	Login ID of the user who performed the logged operation.
IPADDR	VARCHAR(255)	IP Address of the machine from which the logged operation was performed.
LOGGEDBY	VARCHAR(255)	Login ID of the user who writes the log record.
HOSTNAME	VARCHAR(255)	Host name of machine from which the logged operation was performed.
MESSAGEID	VARCHAR(255)	Non-internationalized message identifier for this log record's message.
CONTEXTID	VARCHAR(255)	Identifier associated with a particular login session.

Log File Types: Error and Access

Access log files and error log files are the two types of log files used in OpenSSO Enterprise. *Access log files* record general auditing information concerning the OpenSSO Enterprise deployment. An access log may contain a single record for an event (such as a successful authentication), or multiple records for the same event. For example, when an administrator uses the console to change an attribute value, the Logging Service logs the attempt to change in one record but, it also logs the results of the execution of the change in a second record. *Error log files* record errors that occur within the application. While an operation error is recorded in the error log, the operation attempt is recorded in the access log file.

Flat log files are appended with the `.error` or `.access` extension. Database column names end with `_ERROR` or `_ACCESS`. For example, a flat file logging console events is named `amConsole.access` while a database column logging the same events is named `AMCONSOLE_ACCESS` or `amConsole_access`.

Note – The period (.) separator in a log filename is converted to an underscore (_) in database formats. Also in databases, table names may be converted to all upper case. For example, `amConsole.access` may be converted to `AMCONSOLE_ACCESS`, or it may be converted to `amConsole_access`.

Secure Logging

Secure logging adds an extra measure of security to the Logging Service. When secure logging is enabled, the Logging Service can detect unauthorized changes to the security logs. No special coding is required to leverage this feature. However, secure logging uses a certificate that you must create and install in the container that runs OpenSSO Enterprise. When secure logging is enabled, a Manifest Analysis and Certification (MAC) is generated and stored for every log record, and a special signature record is periodically inserted in the log. The signature record represents the signature for the contents of the log written up to that point. The combination of the certificate and the signature record ensures that the logs have not been tampered. For detailed information about enabling secure logging, see Chapter 14, "Logging Service," in *Sun Federated Access Manager 8.0 Administration Guide*.

Remote Logging

Remote logging allows a client using the Client SDK to create log records on an instance of OpenSSO Enterprise deployed on a remote machine. Remote logging is useful in the following situations:

- When the login URL in the Naming Service of an OpenSSO Enterprise instance points to a remote OpenSSO Enterprise instance, and a trust relationship between the two instances has been configured. (Examples of a trust relationship include servers in a site or using SAML communication.)
- When the OpenSSO Enterprise API are installed in a remote OpenSSO Enterprise instance, and a client application or a simple Java class running on the OpenSSO Enterprise server uses them. A `SSOToken` for the subject of the log records and the identity doing the logging is required. The identity doing the logging must also be authorized to write to the logs.
- When logging APIs are used by OpenSSO Enterprise agents.

OpenSSO Enterprise Component Logs

The log files record a number of events for each of the OpenSSO Enterprise components using the Logging Service. Administrators typically review these log files on a regular basis. Table 5–3 provides a brief description of the log files produced by each OpenSSO Enterprise component.

TABLE 5–3 Access Manager Component Logs

Component	Log Filename	Information Logged
Session Service	■ `amSSO.access`	Session management attributes values such as login time, logout time, and time out limits. Also session creations and terminations.
Administration Console	■ `amConsole.access` ■ `amConsole.error`	User actions performed through the administration console such as creation, deletion and modification of identity-related objects, realms, and policies. `amConsole.access` logs successful console events while `amConsole.error` logs error events.
Authentication Service	■ `amAuthentication.access` ■ `amAuthentication.error`	User logins and log outs, both successful and failed.
Federation Services	■ `amFederation.access` ■ `amFederation.error` ■ `amLiberty.access` ■ `amLiberty.error`	Federation-related events such as the creation of an authentication domain or the creation of a hosted provider entity.
Policy Service (Authorization)	■ `amPolicy.access` ■ `amPolicy.error` ■ `amAuthLog`	Events related to authorization such as policy creation, deletion, or modification, and policy evaluation. `amPolicy.access` logs policy allows, `amPolicy.error` logs policy error events, and `amAuthLog` logs policy denies.
Policy Agents	`amAgent`	Exceptions regarding resources that were either accessed by a user or denied access to a user. `amAgent` logs reside on the server where the policy agent is installed. Agent events are logged on the OpenSSO Enterprise machine in the Authentication logs.
SAML	■ `amSAML.access` ■ `amSAML.error`	**[Remark 5–1 Reviewer: Does this record SAML 1 and 2? If not which version? And where is the one mot recorded here actually recorded?]** SAML-related events such as assertion and artifact creation or removal, response and request details, and SOAP errors.
Command Line	■ `amAdmin.access` ■ `amAdmin.error`	Event successes and errors that occur during operations using the command line tools. Loading a service schema, creating policy, and deleting users are some examples of command line operations.

TABLE 5–3 Access Manager Component Logs *(Continued)*

Component	Log Filename	Information Logged
Password Reset	■ `amPasswordReset.access`	Password reset events.

For detailed reference information about events recorded in each type of OpenSSO Enterprise log, see Chapter 14, "Logging Service," in *Sun Federated Access Manager 8.0 Administration Guide*.

Logging Service Interfaces

There are two Java interfaces provided with the Logging Service. The Java application programming interface (API) `com.sun.identity.log` provides the means for an application external to OpenSSO Enterprise to record events to, and retrieve records from, the Logging Service. `LogRecord` and `Logger` are used for writing, while `LogReader`, `LogQuery`, and `QueryElement` are used for reading. With this API it is possible to write a custom log reading program by setting up queries to retrieve specific records from the log file or database. The Java service provider interface (SPI) `com.sun.identity.log.spi` is used to develop plug-ins to the Logging Service for authorization and other service implementations of secure logging.

[Remark 5–2 Reviewer: Burt, why aren't these in Javadoc? I found this URL: wikis.sun.com/download/attachments/6456057/ITimestampGenerator.html but it's not in my install's Javadoc from last Friday's nightly.] Other pluggable SPI with interface definitions may be used; for example, `ITimestampGenerator` and `SecureTimestampGenerator`. Existing provider modules may be useful as models for writing additional providers. For more information, see the *Federated Access Manager 8.0 Java API Reference* and the *Sun Federated Access Manager 8.0 Developer's Guide*.

Note – OpenSSO Enterprise also has a Logging Service API for C applications. For more information, see the *Sun Federated Access Manager 8.0 C API Reference*.

PART II

Access Control Using OpenSSO Enterprise

User authentication, authorization for access to protected resources, and programmatically defining user sessions are all aspects of access management, one of the core functions of Sun OpenSSO Enterprise. OpenSSO Enterprise offers access management features programmatically using the Client SDK, over the wire using HTTP and the OpenSSO Enterprise console, and using an integrated development environment (IDE) application to incorporate Representational State Transfer (REST) calls and Web Services Definition Language (WSDL) files. The chapters in this part contain information on these aspects of access management.

◆ ◆ ◆ CHAPTER 6

User Sessions and the Session Service

The Session Service in Sun OpenSSO Enterprise tracks a user's interaction with web applications through the use of session data structures, session tokens, cookies, and other objects. This chapter explains these concepts and other components of a user's session and contains the following sections:

About the Session Service

The Session Service in Sun OpenSSO Enterprise tracks a user's interaction with protected web applications. For example, the Session Service maintains information about how long a user has been logged in to a protected application, and enforces timeout limits when necessary. Additionally, the Session Service:

- Generates session identifiers.
- Maintains a master copy of session state information.
- Implements time-dependent behavior of sessions.
- Implements session life cycle events such as logout and session destruction.
- Generates session life cycle event notifications.
- Generates session property change notifications.
- Implements session quota constraints.
- Implements session failover.
- Enables single sign-on and cross-domain single sign-on among applications external to OpenSSO Enterprise.
- Offers remote access to the Session Service through the Client SDK with which user sessions can be validated, updated, and destroyed.

The state of a particular session can be changed by user action or timeout. Figure 6–1 illustrates how a session is created as *invalid* before authentication, how it is activated following a successful authentication, and how it can be invalidated (and destroyed) based on timeout values.

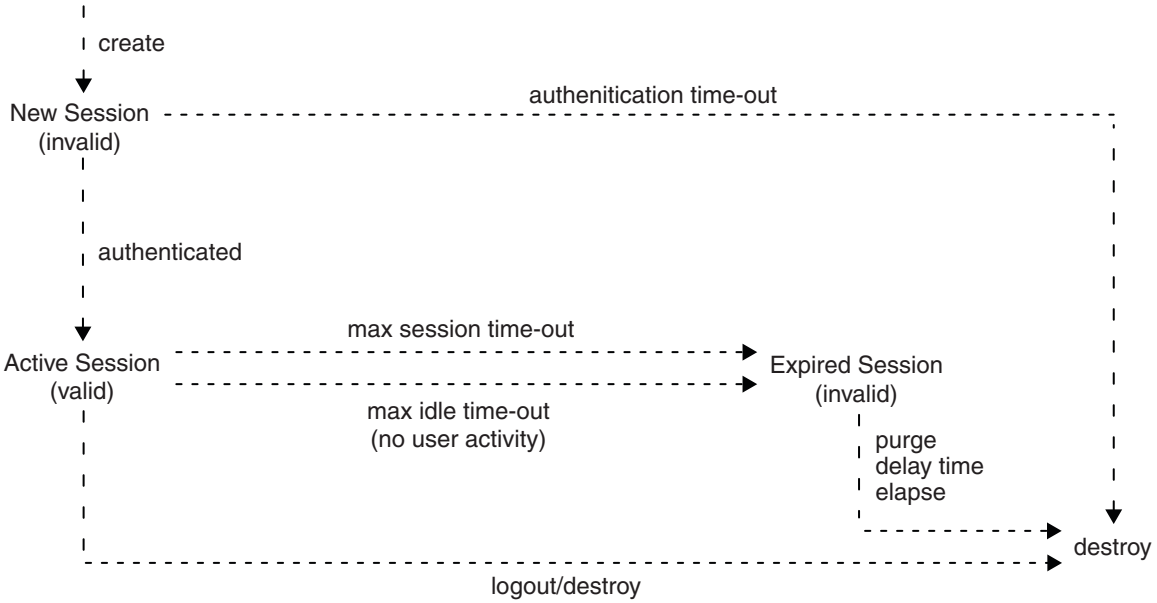


FIGURE 6–1 Life Cycle of a Session

User Sessions and Single Sign-on

A *user session* is the interval between the moment a user attempts to log in to a resource protected by OpenSSO Enterprise, and the moment the session expires, is terminated by an administrator, or the user logs out. As an example of a user session, an employee attempts to access the corporate benefits administration application protected by OpenSSO Enterprise. A new invalid session is created, and the Authentication Service prompts the user for a username and password to verify the user's identity. Following a successful authentication, the Policy Service and policy agent work together to check that the user has the appropriate permissions to access the protected application and allows or denies access based on the outcome.

Oftentimes, in the same user session (without logging out of the corporate benefits application), the same employee might attempt to access a corporate expense reporting application. Because the expense reporting application is also protected by OpenSSO Enterprise, the Session Service provides proof of the user's authentication, and the employee is allowed to access the expense reporting application (based on the outcome of a second authorization check with the Policy

Service). If access is granted, the employee has accessed more than one application in a single user session without having to reauthenticate. This is called *single sign-on* (SSO). When SSO occurs among applications in more than one DNS domain, it is called *cross-domain single sign-on* (CDSSO). For a more detailed overview of a basic user session, an SSO session, and a CDSSO session, see Chapter 7, "Models of the User Session and Single Sign-On Processes."

Session Data Structures and Session Token Identifiers

The Session Service programmatically creates a session data structure to store information about a user session. The result of a successful authentication results in the validation of a *session data structure* for the user or entity and the creation of a *session token* identifier. The session data structure minimally stores the following information.

Identifier	A unique, universal identifier for the session data structure.
Host Name or IP Address	The location from which the client (browser) is making the request.
Principal	Set to the user's distinguished name (DN) or the application's principal name.
Type	USER or APPLICATION
Session State	Defines whether the session is valid or invalid.
Maximum Idle Time	Maximum number of minutes without activity before the session will expire and the user must reauthenticate.
Maximum Session Time	Maximum number of minutes (activity or no activity) before the session expires and the user must reauthenticate.
Maximum Caching Time	Maximum number of minutes before the client contacts Access Manager to refresh cached session information.

A session can also contain additional properties which can be used by other applications. For example, a session data structure can store information about a user's identity, or about a user's browser preferences. You can configure OpenSSO Enterprise to include the following types of data in a session:

- Protected properties are only modifiable by the server-side modules (primarily the Authentication Service).
- Custom properties are modifiable remotely by any application which possesses the session identifier.

For a detailed summary of information that can be included in a session, see Chapter 10, "Configuring OpenSSO Enterprise Sessions," in *Sun OpenSSO Enterprise 8.0 Installation and Configuration Guide*.

The session token, also referred to as a *sessionID* and programmatically as an `SSOToken`, is an encrypted, unique string that identifies the session data structure. As the user visits different protected resources using the browser, the session token is propagated to these resources and is used to retrieve the user's credentials. These credentials are then validated by sending a back-end request (using the Client SDK or a policy agent) to OpenSSO Enterprise which then returns an error or the session's prior authentication data. Sessions (and hence the `SSOToken`) are invalidated when a user logs out, the session expires, or a user in an administrative role invalidates it. With OpenSSO Enterprise, a session token is carried in a *cookie*, an information packet generated by a web server and passed to a web browser. (The generation of a cookie for a user by a web server does not guarantee that the user is allowed access to protected resources. The cookie simply points to information in a data store from which an access decision can be made.)

Note – Access to some OpenSSO Enterprise services, such as the Policy Service and the Logging Service, require presentation of both the `SSOToken` of the application as well as the `SSOToken` of the user, allowing only designated applications to access these services.

◆ ◆ ◆ CHAPTER 7

Models of the User Session and Single Sign-On Processes

This chapter traces events in a basic user session, a single sign-on session (SSO), a cross-domain single sign-on session (CDSSO), and session termination to give you an overview of the features and processes being invoked by OpenSSO Enterprise. It contains the following sections:

Basic User Session

The following sections describe the process behind a basic user session by tracing what happens when a user logs in to a resource protected by OpenSSO Enterprise. In these examples, the server which hosts an application is protected by a policy agent. The Basic User Session includes the following phases:

Initial HTTP Request

When a user initiates a user session by using a browser to access and log in to a protected web-based application, the events illustrated in Figure 7–1 occur. The accompanying text describes the model.

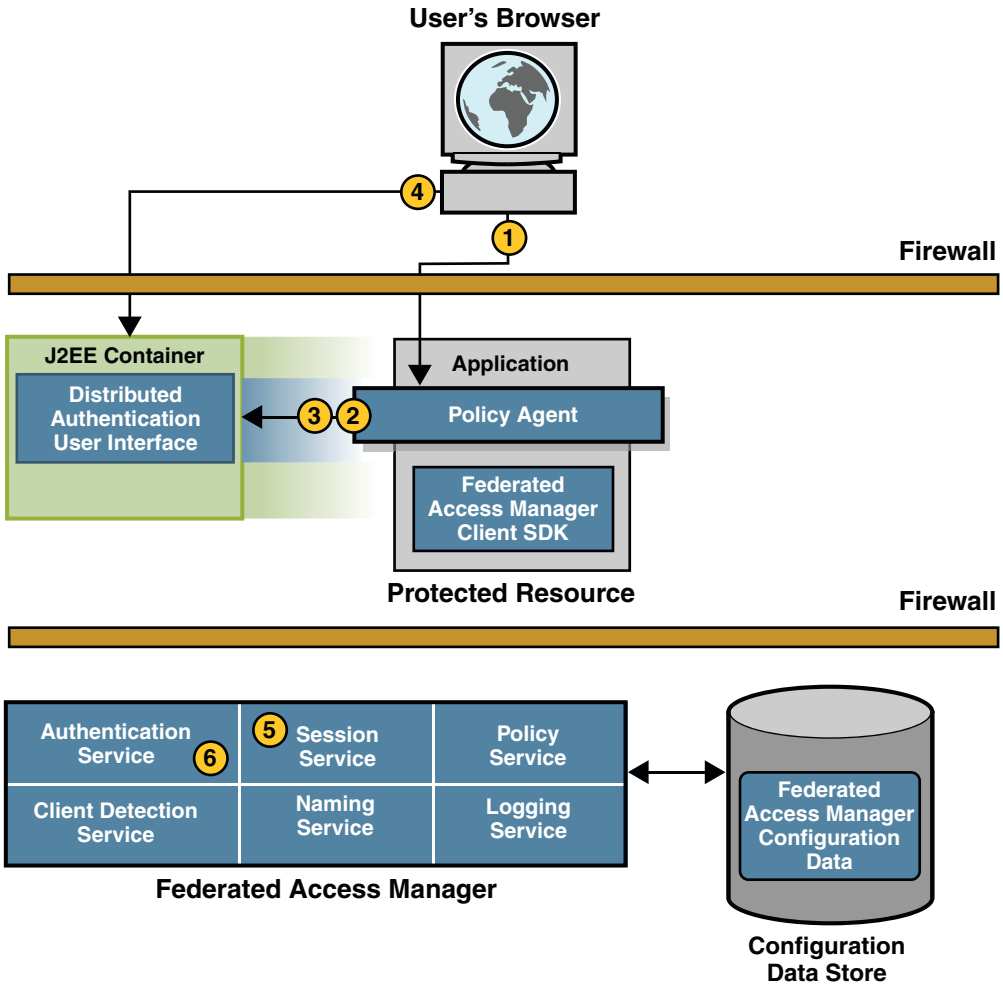


FIGURE 7–1 Initial HTTP Request

1. The user's browser sends an HTTP request to the protected resource.
2. The policy agent that protects the resource intercepts and inspects the user's request and finds no session token.
3. The policy agent issues a redirect to its configured authentication URL to begin the authentication process.

   In this example, the authentication URL it is set to the URL of the Distributed Authentication User Interface.
4. The browser, following the redirect, sends a GET request for authentication credentials to the Distributed Authentication User Interface.

5. The Session Service creates a new session (session data structure) and generates a session token (a randomly-generated string that identifies the session).
6. The Authentication Service sets the session token in a cookie.

The next part of the user session is "User Authentication" on page 93.

User Authentication

When the browser sends the GET request to the Distributed Authentication User Interface, the events illustrated in Figure 7–2 occur.

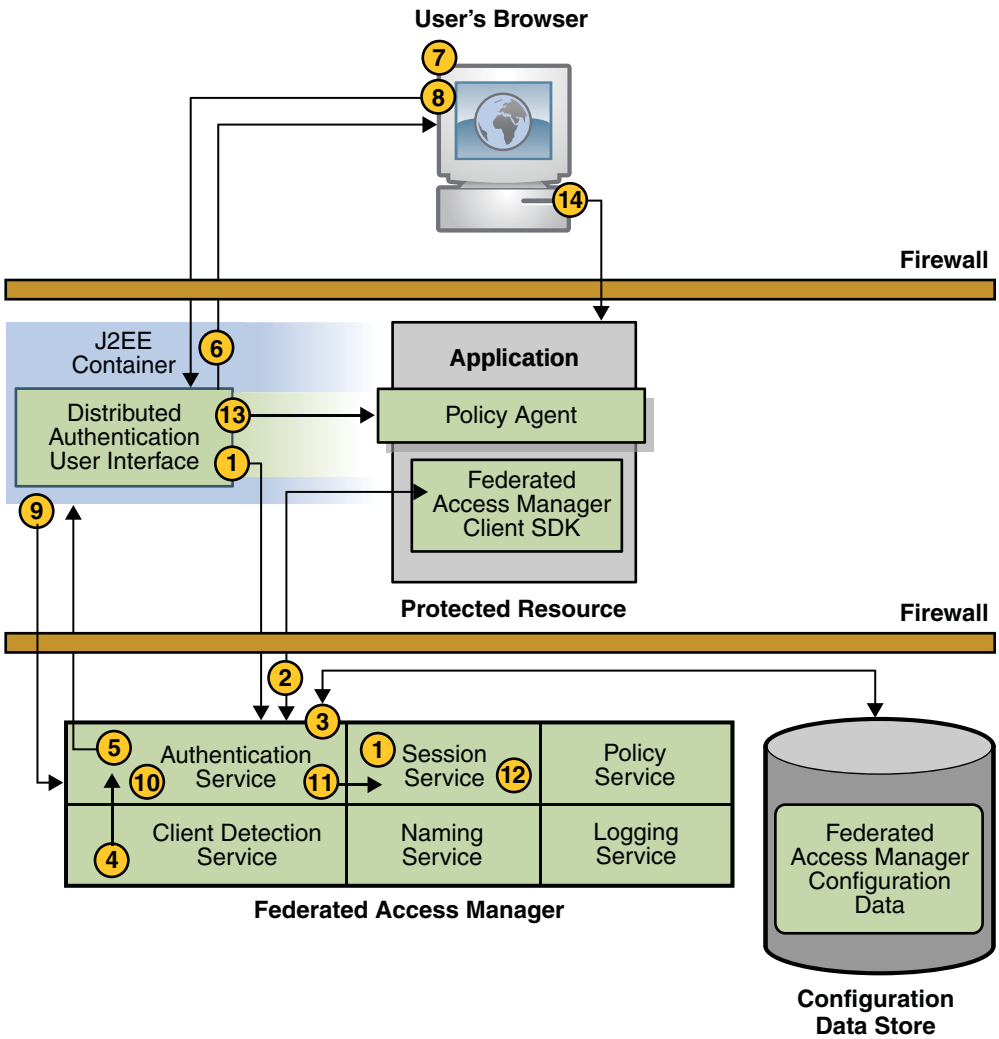


FIGURE 7–2 User Authentication

1. Using the parameters in the GET request, the Distributed Authentication User Interface contacts the OpenSSO Enterprise Authentication Service (which, in turn, communicates with the Session Service).
2. The Authentication Service determines what should be presented to the user based upon configuration data and retrieves the appropriate authentication module(s) and callback(s) information.

   For example, if configured to use LDAP Authentication, the Authentication Service determines that the LDAP Authentication login page should be displayed.

3. The collected information is passed to the Distributed Authentication User Interface using the Client SDK.
4. The Client Detection Service determines which protocol, such as HTML or WML, to use to display the login page.
5. The Distributed Authentication User Interface generates a dynamic presentation extraction page that contains the appropriate credentials request and callbacks information obtained from OpenSSO Enterprise.

   The session cookie will be included in this communication.
6. The user's browser displays the login page.
7. The user enters information in the fields of the login page.
8. The browser sends the credentials in an HTTP POST to the Distributed Authentication User Interface.
9. The Distributed Authentication User Interface uses the Client SDK to pass the credentials to the Authentication Service.
10. The Authentication Service uses the appropriate authentication module to validate the user's credentials.

    For example, if LDAP authentication is used, the LDAP authentication module verifies that the username and password provided exist in the LDAP directory.
11. Assuming authentication is successful, the Authentication Service activates the session by calling the appropriate methods in the Session Service.

    The Authentication Service stores information such as login time, Authentication Scheme, and Authentication Level in the session data structure.
12. Once the session is activated, the Session Service changes the state of the session token to valid.
13. The Distributed Authentication User Interface replies to the protected resource with the validated `SSOToken` in a set-cookie header.
14. Now, the browser makes a second request to the original resource protected by a policy agent.

    This time, the request includes a valid session token created during the authentication process.

The next part of the user session is "Session Validation" on page 95.

Session Validation

After successful authentication, the user's browser redirects the initial HTTP request to the server a second time for validation. The request now contains a session token in the same DNS domain as OpenSSO Enterprise. The events in Figure 7–3 illustrate this process.

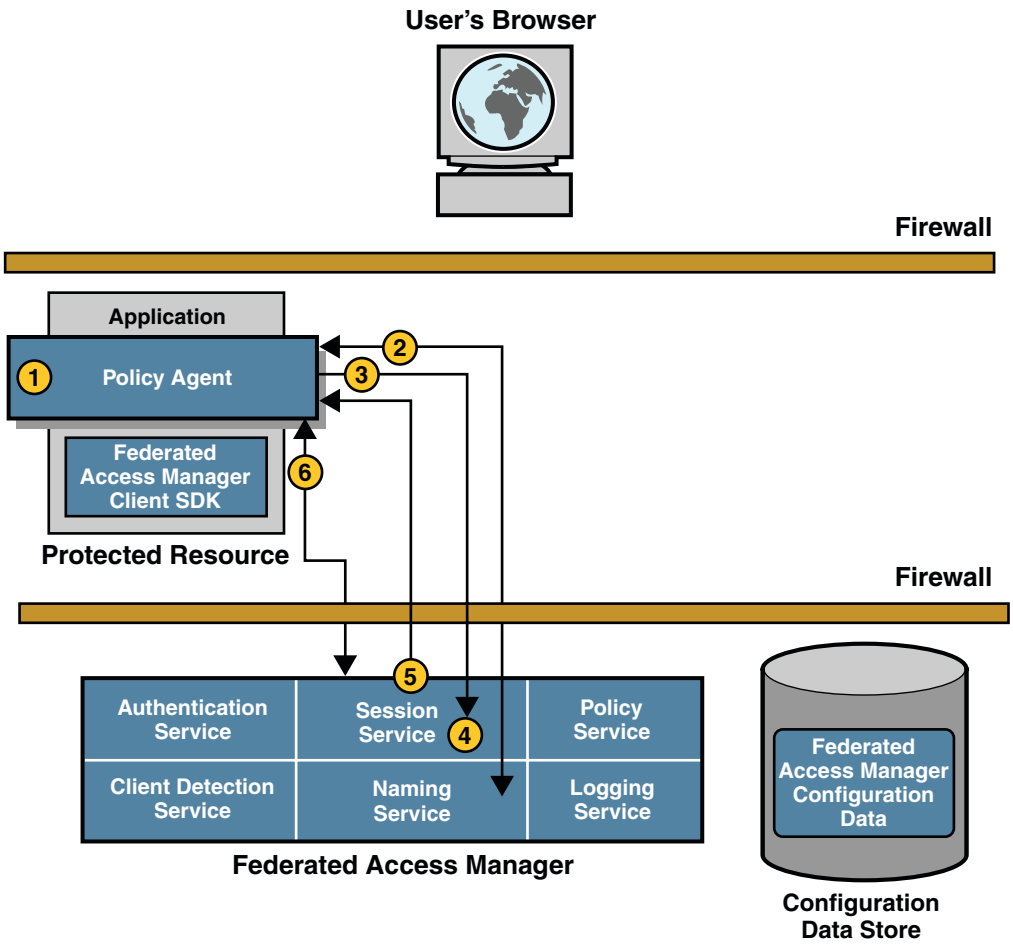


FIGURE 7–3 Session Validation

1. The policy agent intercepts the second access request.
2. To determine the validity of the session token, the policy agent contacts the Naming Service to learn where the session token originated.

   The Naming Service allows clients to find the URL for internal OpenSSO Enterprise services. When contacted, the Naming Service decrypts the session token and returns the corresponding URL which can be used by other services to obtain information about the user session.
3. The policy agent, using the information provided by the Naming Service, makes a POST request to the Session Service to validate the included session token.

4. The Session Service receives the request and determines whether the session token is valid based on the following criteria:
   a. Has the user been authenticated?
   b. Does a session data structure associated with the session token exist?
5. If all criteria are met, the Session Service responds that the session token is valid.

   This assertion is coupled with supporting information about the user session itself.
6. The policy agent creates a Session Listener and registers it with the Session Service, enabling notification to be sent to the policy agent when a change in the session token state or validity occurs.

The next part of the user session is "Policy Evaluation and Enforcement" on page 97.

Policy Evaluation and Enforcement

After a session token has been validated, the policy agent determines if the user can be granted access to the server by evaluating its defined policies. Figure 7–4 illustrates this process.

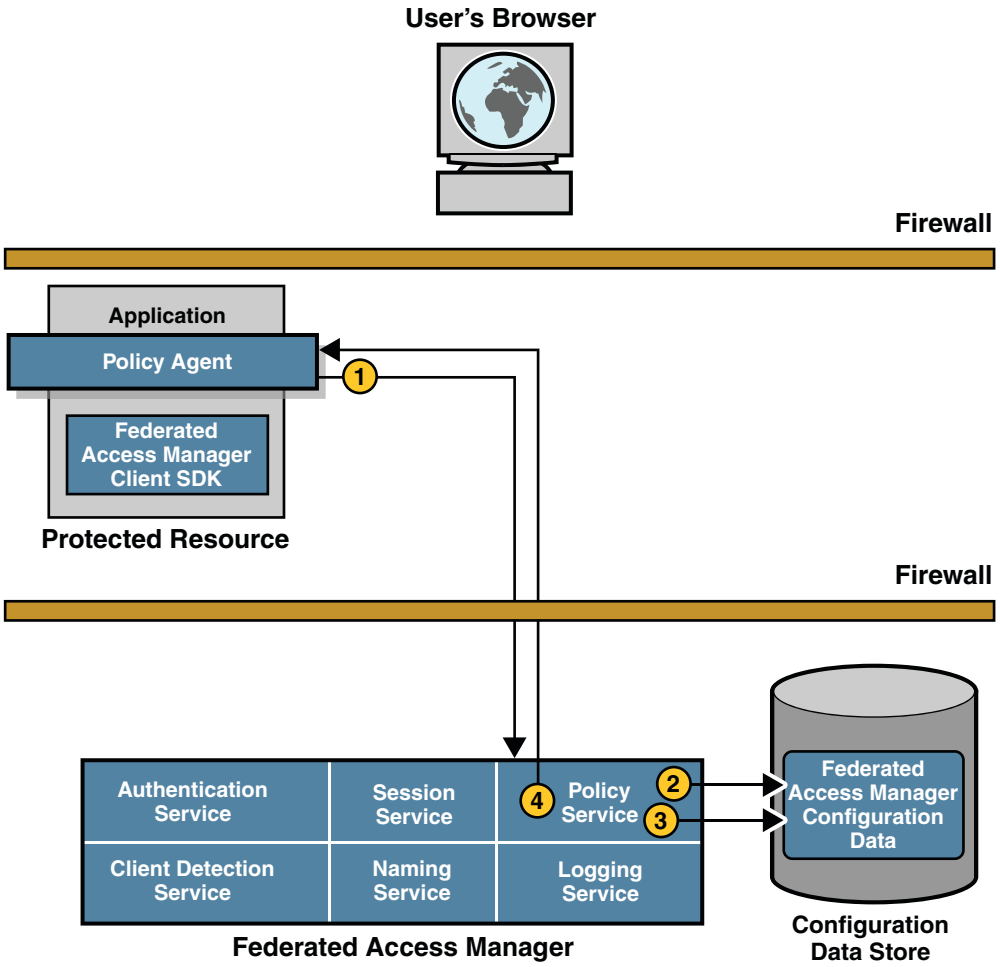


FIGURE 7–4 Policy Evaluation

1. The policy agent sends a request to the Policy Service, asking for decisions regarding resources in its portion of the HTTP namespace.

   The request also includes additional environmental information. For example, IP address or DNS name could be included in the request because they might impact conditions set on a configuration policy.

2. The Policy Service checks for policies that apply to the request.

   Policies are cached in OpenSSO Enterprise. If the policies have not been cached already, they are loaded from OpenSSO Enterprise.

3. If policies that apply to the request are found, the Policy Service checks if the user identified by the session token is a member of any of the Policy Subjects.

a. If no policies that match the resource are found, the user will be denied access.

b. If policies are found that match the resource, and the user is a valid subject, the Policy Service evaluates the conditions of each policy. For example, *Is it the right time of day?* or *Are requests coming from the correct network?*

   - If the conditions are met, the policy applies.
   - If the conditions are not met, the policy is skipped.

4. The Policy Service aggregates all policies that apply, encodes a final decision to grant or deny access, and responds to the policy agent.

The next part of the basic user session is "Logging the Results" on page 99.

Logging the Results

When the policy agent receives a decision from the Policy Service, the events illustrated in Figure 7–5 occur.

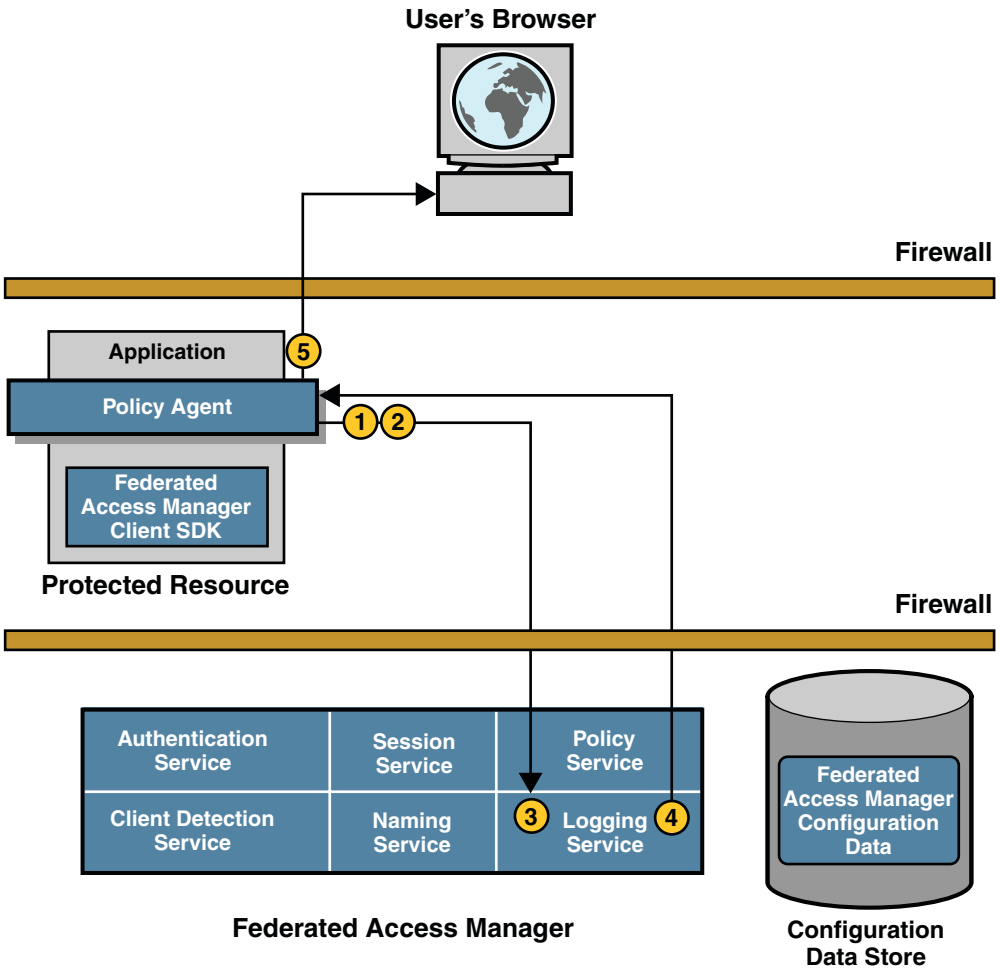


FIGURE 7–5 Logging the Policy Evaluation Results

1. The decision and session token are cached by the policy agent so subsequent requests can be checked using the cache (without contacting OpenSSO Enterprise).

   The cache will expire after a (configurable) interval has passed or upon explicit notification of a change in policy or session status.
2. The policy agent issues a logging request to the Logging Service.
3. The Logging Service logs the policy evaluation results to a flat file (which can be signed) or to a JDBC store, depending upon the log configuration.
4. The Logging Service notifies the policy agent of the new log.
5. The policy agent allows or denies the user access to the application.

a. If the user is denied access, the policy agent displays an "access denied" page.

b. If the user is granted access, the resource displays its access page.

Assuming the browser displays the application interface, this basic user session is valid until it is terminated. See "Session Termination" on page 105 for more information. While logged in, if the user attempts to log into another protected resource, the "Single Sign-On Session" on page 101 begins.

Single Sign-On Session

Remark 7–1 Reviewer **There are 13 bullets in graphic and nine steps? Huh????? What should be changed? From email with subject: Auth/SSO process question not answered. check against new graphic**

SSO is always preceded by a basic user session in which a session is created, its session token is validated, the user is authenticated, and access is allowed. SSO begins when the authenticated user requests a protected resource on a second server in the **same** DNS domain. The following process describes an SSO session by tracking what happens when an authenticated user accesses a second application in the same DNS domain as the first application. Because the Session Service maintains user session information with input from all applications participating in an SSO session, in this example, it maintains information from the application the user was granted access to in "Basic User Session" on page 91.

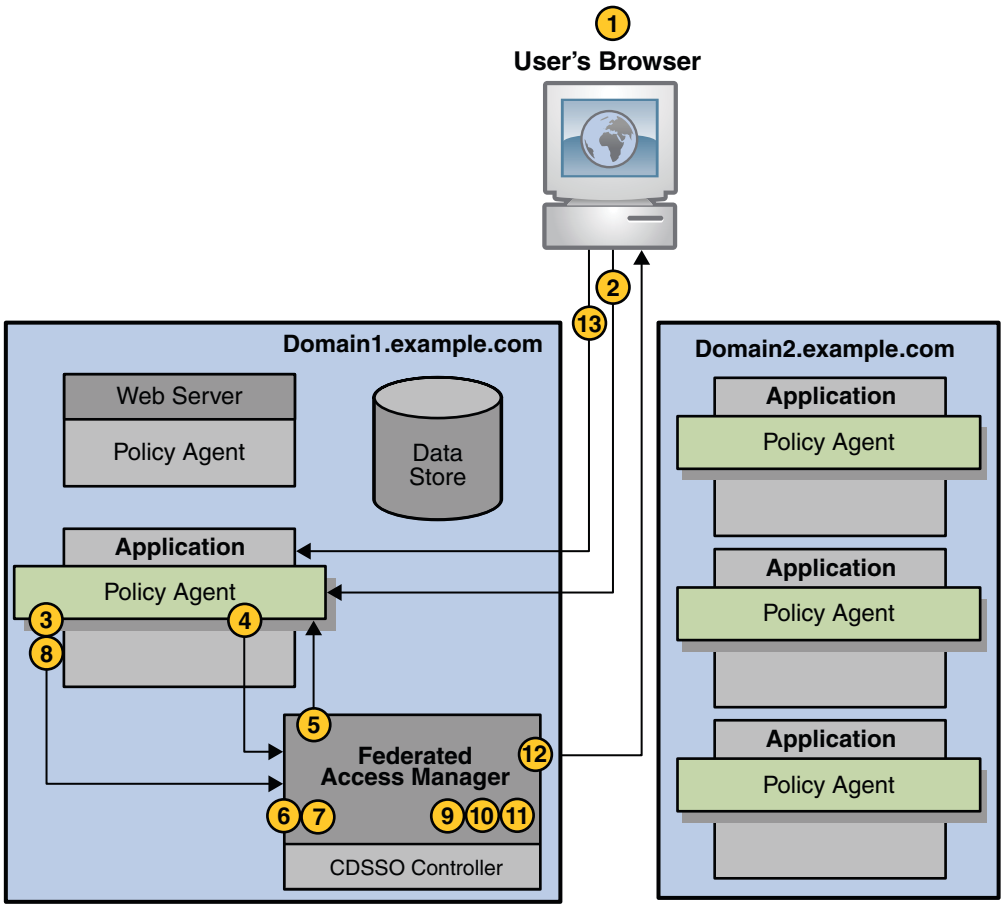


FIGURE 7–6 Single Sign-On Session

1. The user attempts to access a second application hosted on a server in the same domain as the first application to which authentication was successful.
2. The user's browser sends an HTTP request to the second application that includes the user's session token.
3. The policy agent intercepts the request and inspects it to determine whether a session token exists.

   A session token indicates the user is already authenticated. Since the user was authenticated, the Authentication Service is not required at this time. The Session Service API retrieve the session data using the session token identifier imbedded in the cookie.
4. The policy agent sends the session token identifier to the OpenSSO Enterprise Session Service to determine whether the session is valid or not.

For detailed steps, see "Session Validation" on page 95.

5. The Session Service sends a reply to the policy agent indicating whether the session is valid.
   - If the session is not valid, the user is redirected to the Authentication page.
   - If the session is valid, the Session Service creates a Session Listener.
6. As the session is valid, the Session Service creates a Session Listener.

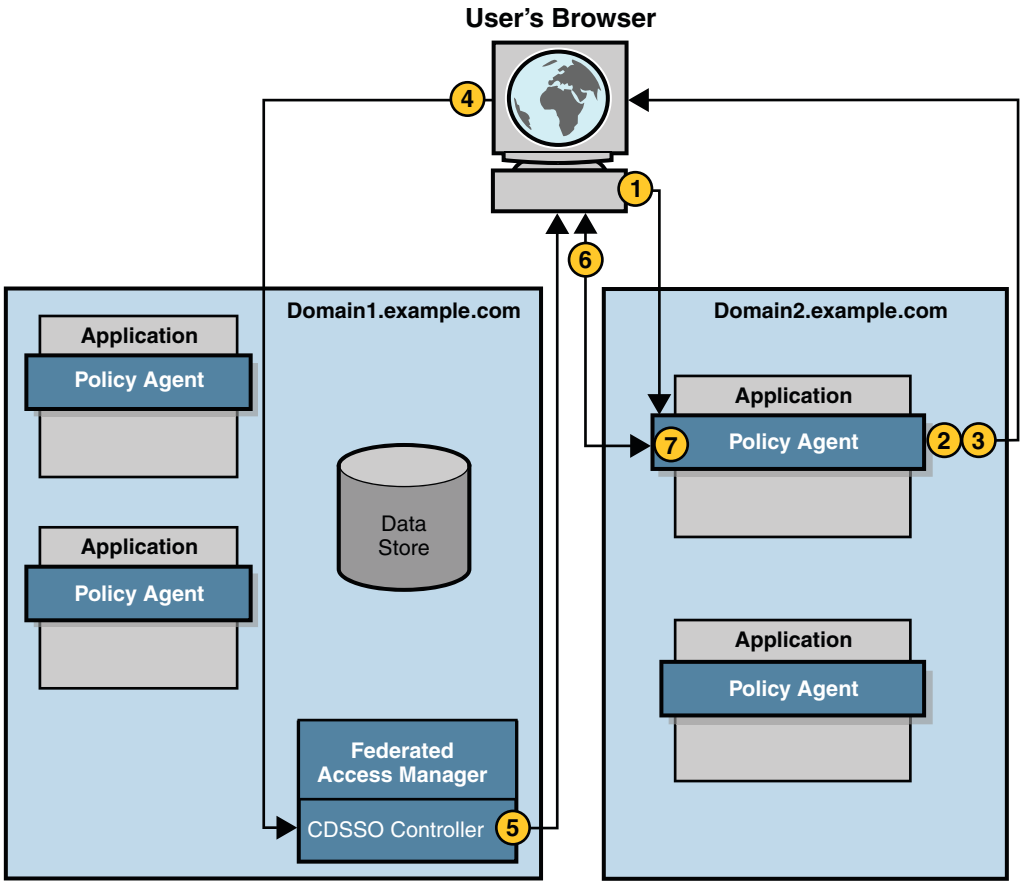
   A Session Listener sends notification to the policy agent when a change in the session state occurs.
7. The policy agent sends a request for a decision regarding resources in it's portion of the HTTP namespace to the Policy Service.
8. The Policy Service checks for policies that apply to the request.
9. The Policy Service sends the policy evaluation response (either Access Denied or Access Granted.) to the policy agent.
   - If Policy Service does not find policy allowing access to the protected resource, the user is denied access and the Logging Service logs a denial of access. The user may be redirected to a specified page indicating that access was denied if configured as such by the administrator.
   - If the Policy Service finds policy allowing access to the protected resource, the user is granted access and the session is valid until terminated.
10. The policy agent sends a reply to the user indicating whether the user is granted the access.
    - If the user is denied access, the policy agent displays an Access Denied page.
    - If the user is granted access, the protected resource displays its access page.

Assuming the Policy Service finds policy allowing access to the protected resource, the user is granted access and the SSO session is valid until terminated. See "Session Termination" on page 105. While still logged in, if the user attempts to log in to another protected resource located in a different DNS domain, the "Cross-Domain Single Sign-On Session" on page 103 begins.

Cross-Domain Single Sign-On Session

CDSSO occurs when an authenticated user requests a protected resource on a server in a **different** DNS domain. The user in the previous sections, "Basic User Session" on page 91 and "Single Sign-On Session" on page 101, for example, accessed applications in one DNS domain. In this scenario, the CDSSO Controller within OpenSSO Enterprise transfers the user's session information from the initial domain, making it available to applications in a second domain.

Note – The basic difference between the proprietary CDSSO and SAML v2 (as described in "Federation Options" on page 155) is that CDSSO uses a single authentication authority, a mechanism to move a cookie between multiple DNS domains. SAML v2, on the other hand, gives you the option of using multiple authentication authorities, with one authority asserting the identity of the user to the other.



1. The authenticated user's browser sends an HTTP request to the application in a different DNS domain.
2. The policy agent intercepts the request and inspects it to determine if a session token exists for the domain *in which the requested application exists*. One of the following occurs:
   - If a session token is present, the policy agent validates the session.

- If no session token is present, the policy agent (which is configured for CDSSO) will redirect the HTTP request to the CDSSO Controller.

  The CDSSO Controller uses Liberty Alliance Project protocols to transfer sessions so the relevant parameters are included in the redirect.

In this example, no session token for the second domain is found.

3. The policy agent redirects the HTTP request to the CDSSO Controller.
4. The user's browser allows the redirect to the CDSSO Controller.

   Recall that earlier in the user session the session token was set in a cookie in the first domain which is now part of the redirect.
5. The CDC Servlet (in the CDSSO Controller) receives the session token from the first domain, extracts the user's session information, formulates a Liberty POST profile response containing the information, and returns a response to the browser.
6. The user's browser automatically submits the response to the policy agent in the second domain.

   The POST is based upon the Action and the Javascript included in the Body tags `onLoad`.
7. The policy agent in the second domain receives the response, extracts the session information, validates the session, and sets a session token in the cookie for the new DNS domain.

The process continues with "Policy Evaluation and Enforcement" on page 97 and "Logging the Results" on page 99. Based on the policy outcome, the user is granted or denied access to the application.

1. If the user is denied access, the policy agent displays an "access denied" page.
2. If the user is granted access, the protected resource displays its access page. The new cookie can now be used by all agents in the new domain, and the session is valid until it is terminated.

Session Termination

A user session can be terminated in any of following ways:

User Ends Session

When a user explicitly logs out of OpenSSO Enterprise by clicking on a link to the Logout Service the following events occur:

1. The Logout Service receives the Logout request, and:
   a. Marks the user's session as destroyed.
   b. Destroys the session.
   c. Returns a successful logout page to the user.
2. The Session Service notifies applications which are configured to interact with the session. In this case, each of the policy agents was configured for Session Notification, and each is sent a document instructing the agent that the session is now invalid.
3. The policy agents flush the session from the cache and the user session ends.

Administrator Ends Session

OpenSSO Enterprise administrators with appropriate permissions can terminate a user session at any time. When an administrator uses the Sessions tab in the OpenSSO Enterprise console to end a user's session, the following events occur:

1. The Logout Service receives the Logout request, and:
   a. Marks the user's session as destroyed.
   b. Destroys the session.
2. The Session Service notifies applications which are configured to interact with the session. In this case, each of the policy agents was configured for Session Notification, and each is sent a document instructing the agent that the session is now invalid.
3. The policy agents flush the session from cache and the user session ends.

OpenSSO Enterprise Enforces Timeout Rules

When a session timeout limit is reached, the Session Service:

1. Changes the session status to `invalid`.
2. Displays a time out message to the user.
3. Starts the timer for purge operation delay. (The default is 60 minutes.)
4. Purges or destroys the session when the purge operation delay time is reached.
5. Displays login page to the user if a session validation request comes in after the purge delay time is reached.

Session Quota Constraints

OpenSSO Enterprise allows administrators to constrain the amount of sessions one user can have. If the user has more sessions than the administrator will allow, one (or more) of the existing sessions can be destroyed.

◆ ◆ ◆ CHAPTER 8

Authentication and the Authentication Service

The Sun OpenSSO Enterprise Authentication Service determines whether a user is the person he claims to be. User authentication is the first step in controlling access to web resources within an enterprise. This chapter explains how the Authentication Service works with other components to prove that the user's identity is genuine. Topics covered in this chapter include:

Authentication Service Overview

The function of the Authentication Service is to request information from an authenticating party, and validate it against the configured identity repository using the specified authentication module. After successful authentication, the user session is activated and can be validated across all web applications participating in an SSO environment. For example, when a user or application attempts to access a protected resource, credentials are requested by one (or more) authentication modules. Gaining access to the resource requires that the user or application be allowed based on the submitted credentials. From the user perspective, a company employee wants to look up a colleague's phone number. The employee uses a browser to access the company's online phone book. To log in to the phone book service, the employee provides a user name and password. OpenSSO Enterprise compares the user's input with data stored in the appropriate identity repository. If OpenSSO Enterprise finds a match for the user name, and if the given password matches the stored password, the user's identity is authenticated.

Note – The "Basic User Session" on page 91 section in the previous chapter contains a detailed description of the authentication process itself.

The Authentication Service can be accessed by a user with a web browser, by an application using the Client SDK, or by any other client that correctly implements the Authentication Service messaging interfaces. The Authentication Service framework has a pluggable architecture for authentication modules that have different user credential requirements. Together with the Session Service, the Authentication Service establishes the fundamental infrastructure for SSO. Generally speaking, the Authentication Service:

- Identifies a requester's credential requirements.
- Generates a dynamic user interface based on the requirements of the authentication module being called.
- Supports custom, pluggable authentication modules.
- Provides pre- and post-processing SPI.
- Populates and manages system domain cookies.
- Generates time dependent alerts and session termination notifications.
- Provides a remote user interface application for distributed deployments.
- Implements a clean logout interface which destroys the session.

Every time a request is used to access the Authentication Service, the session token identifier is retrieved and used to get the associated session data structure from the Session Service. Additionally, the Authentication Service interfaces with the Session Service to:

- Initiate or create user sessions.
- Maintain session state information.
- Activate sessions after successful authentication.
- Populate the valid session data structure with all user-authenticated identity data and properties.
- Destroy sessions after logout.

The following diagram illustrates how the two services work together.

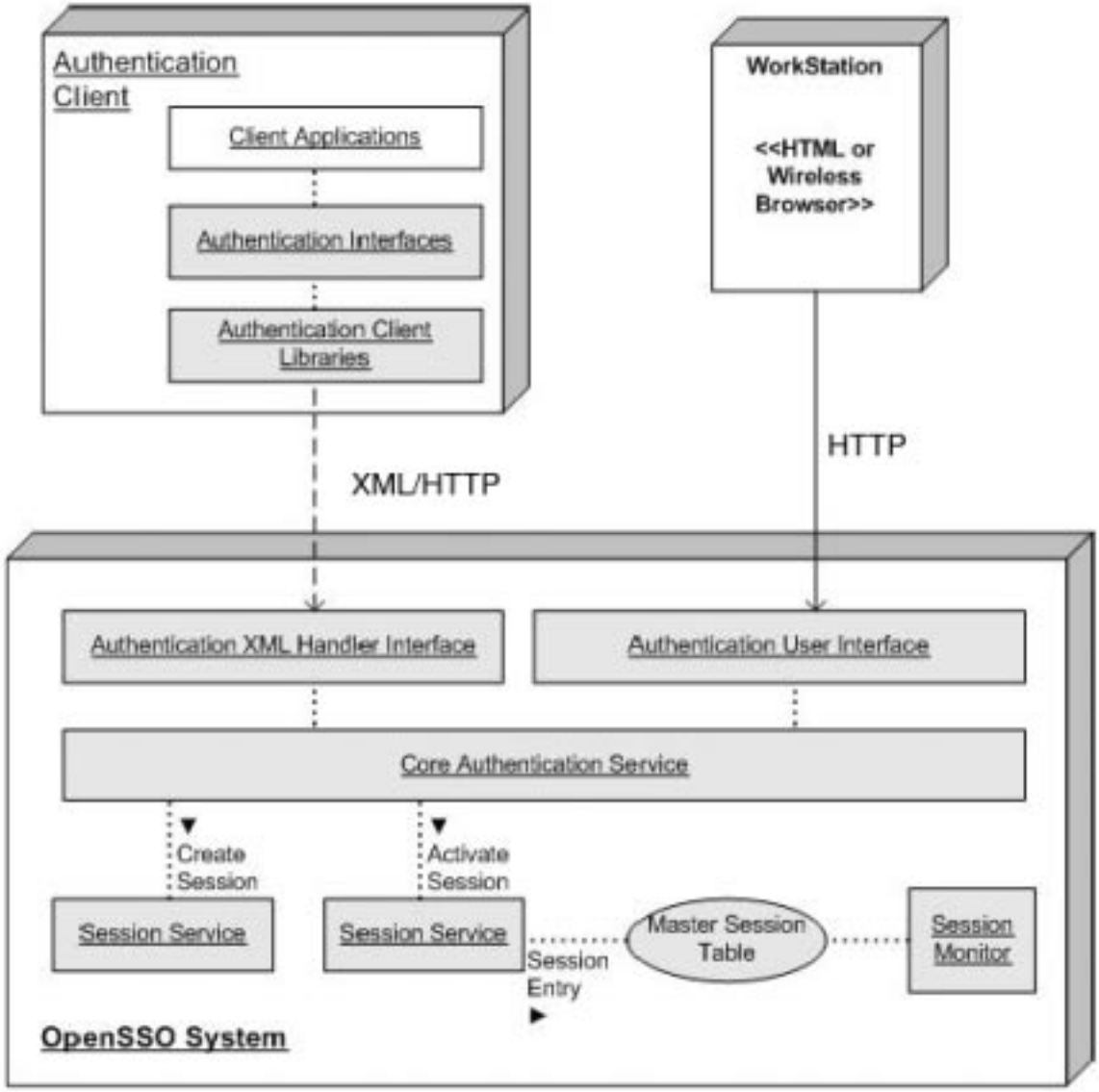


FIGURE 8–1 Authentication Service and Session Service Interfaces

The Authentication Service also interfaces with other OpenSSO Enterprise services including the Naming Service, the Identity Repository Service, the Logging Service, and the Monitoring Service. It also interfaces with the configuration data store and policy agents protecting system resources. (A policy agent must authenticate itself using the Client SDK authentication interfaces, and users with no valid session must be authenticated.)

Authentication Service Features

The following sections explain some of the features of the Authentication Service.

Client Detection

Because the Authentication Service is client-type aware, the initial step in the authentication process is to identify the type of client making the HTTP(s) request. This feature is known as *client detection*. The URL information in the HTTP(s) request is used to retrieve the client's characteristics. Based on these characteristics, the appropriate authentication pages are returned. For example, when a web browser is used for requesting access, an HTML login page will be displayed. Once the user is authenticated, the client type is added to the session token for future use. For more information, see Chapter 11, "Identifying the Client Type," in *Sun Federated Access Manager 8.0 Developer's Guide*.

Note – OpenSSO Enterprise supports all configured client types including cookie-less and cookie-enabled. Currently, there are some restrictions to mobile client detection.

Account Locking

The Authentication Service provides account locking to prevent a user from completing the authentication process after a specified number of failures. OpenSSO Enterprise sends email notifications to administrators when account lockouts occur. OpenSSO Enterprise supports:

Physical Locking. By default, user accounts are physically unlocked. You can initiate physical locking by typing `inactive` as the value of the Lockout Attribute Name attribute in the Core Authentication Service. Additionally, the value of the Login Failure Lockout Duration attribute should be set to `0`.

physical lock attr name is: inetuserstatus value active/inactive

Memory Locking. You can configure Memory Locking so that a user account is locked in memory after a specified number of authentication attempts. By changing the Login Failure Lockout Duration attribute to a value greater

then `0`, the user's account is then locked in memory for the number of minutes specified and the account is unlocked after the time period elapses.

To figure out the amount of time the lockout will be in effect, the value of the Lockout Duration Multiplier attribute is multiplied by the value of the Login Failure Lockout Duration attribute for subsequent lockout. For example, if the value of Login Failure Lockout Duration is 5 minutes and the value of the Lockout Duration Multiplier is 2, the first time a user is locked out in memory will be 5 minutes. The second time this same user gets locked out in memory the lockout duration will be 10 minutes (5 minutes x 2). The third time this same user gets locked out in memory the lockout duration will be 20 minutes (5 minutes x 2 x 2).

The account locking feature is disabled by default. Account locking activities are also logged. For information on how to enable it, see "Account Locking" in *Sun Federated Access Manager 8.0 Administration Guide*.

Note – Only authentication modules that throw an `Invalid Password Exception` can leverage the Account Locking feature. Out of the box, these include Active Directory, Data Store, HTTP Basic, LDAP, and Membership.

Authentication Chaining

OpenSSO Enterprise allows the configuration of an authentication process in which a user must pass credentials to one or more authentication modules before session validation is accomplished. This is called *authentication chaining*. OpenSSO Enterprise uses the Java Authentication and Authorization Service (JAAS) framework (integrated with the Authentication Service) to implement authentication chaining. The JAAS framework validates all user identifiers used during the authentication process, and maps them to one principal. (The mapping is based on the configuration of the User Alias List attribute in the user's profile.) If all the maps are correct, the session token is validated. If all the maps are not correct, the user is denied a valid session token. Once authentication to all modules in the chain succeeds or fails, control is returned to the Authentication Service from the JAAS framework.

You configure an authentication chain by realm, user, role, or service. Determining validation is based upon the control flag configured for each authentication module instance defined in the chain. The flags are:

Requisite — Authentication to this module instance is required to succeed. If it succeeds, authentication continues down the module instance list. If it fails, control immediately returns to the application.

Required	Authentication to this module instance is required to succeed. If any of the required module instances defined in the chain fails, the whole authentication chain will fail.
Sufficient	The module instance is not required to succeed. If it does succeed, control immediately returns to the application (authentication does not proceed down the module instance list). If it fails, authentication continues down the list.
Optional	The module instance is not required to succeed. Whether it succeeds or fails, authentication still continues to proceed down the module instance list.

Note – Role based authentication is only supported for use with the AM SDK data store. This data store would come from an existing Sun Java System Access Manager 7.x installation or would have been manually created.

For more information, see "Authentication Modules and Chains" in *Sun Federated Access Manager 8.0 Administration Guide*. For an overview of the authentication module instances, see "Authentication Modules" on page 116.

Fully Qualified Domain Name Mapping

Fully Qualified Domain Name (FQDN) mapping enables the Authentication Service to take corrective action in the case where a user may have typed in an incorrect URL. This is necessary, for example, when a user specifies a partial host name or IP address to access protected resources. This feature can also be used to allow access to one instance of OpenSSO Enterprise using many different aliases. For example, you might configure one instance of OpenSSO Enterprise as `intranet.example.com` for employees and `extranet.example.com` for partners. For more information, see "Fully Qualified Domain Name Mapping" in *Sun Federated Access Manager 8.0 Administration Guide*.

Persistent Cookies

A *persistent cookie* is an information packet that is written to the user's hard drive and, therefore, continues to exist after the web browser is closed. The persistent cookie enables a user to log into a new browser session without having to reauthenticate. The Authentication Service can be enabled to write persistent cookies rather than cookies that are written to a web browser's memory. For more information, see "Persistent Cookie" in *Sun Federated Access Manager 8.0 Administration Guide*.

Session Upgrade

The Authentication Service allows for the upgrade of a valid session based on a second, successful authentication performed by the same user. If a user with a valid session attempts to authenticate to a second resource secured under the realm to which he is currently authenticated, and this second authentication request is successful, the Authentication Service updates the session with the new properties based on the new authentication. If the authentication fails, the current user session is returned without an upgrade. If the user with a valid session attempts to authenticate to a resource secured in a different realm, the user will receive a message asking whether the user would like to authenticate to the new realm. The user can choose to maintain the current session, or can attempt to authenticate to the new realm. Successful authentication will result in the old session being destroyed and a new one being created. For more information, see "Session Upgrade" in *Sun Federated Access Manager 8.0 Administration Guide*.

Note – Successful authentication for session upgrade does not necessarily destroy the previous session. If the subsequent `AuthContext` object is created with the constructor `AuthContext(SSOToken ssoToken, boolean forceAuth)` when `forceAuth` is set to `true`, the existing session will be used and a new session will not be created.

JAAS Shared State

The JAAS shared state enables sharing of both a user identifier and a password between authentication module instances. Options are defined for each authentication module type by realm, user, service and role. If an authentication fails with the credentials from the shared state, the authentication module restarts the authentication process by prompting for its required credentials. If it fails again, the module is marked failed. After a commit, an abort, or a logout, the shared state will be cleared. For more information, see "JAAS Shared State" in *Sun Federated Access Manager 8.0 Administration Guide*.

Security

From a security point of view, here are some general practices implemented in the Authentication Service.

- SSL is strongly recommended to prevent the user credentials from being stolen through passive network snooping.
- The signing and encryption of some user data is to prevent other software applications, sharing the same system resources, from subverting it.

- The main user entry points of the Authentication Service (Distributed Authentication User Interface, Authentication XML Handler Interface for remote clients, the Authentication Service User Interface) are protected by entry level validation of the size of the requested data.
- Creation and modification of authentication configuration information is only allowed by privileged OpenSSO Enterprise administrators.

Authentication Modules

An *authentication module* is a plug-in that collects user information such as a user ID and password, and compares the information against entries in a database. If a user provides information that meets the authentication criteria, the user is validated and, assuming the appropriate policy configuration, granted access to the requested resource. If the user provides information that does not meet the authentication criteria, the user is not validated and denied access to the requested resource. OpenSSO Enterprise is deployed with a number of authentication modules. Table 8–1 provides a brief description of each.

TABLE 8–1 Authentication Service Modules

Authentication Module Name	Description
Active Directory	Uses an Active Directory operation to associate a user identifier and password with a particular Active Directory entry. You can define multiple Active Directory authentication configurations for a realm. Allows both LDAP and Active Directory to coexist under the same realm.
Anonymous	Enables a user to log in without specifying credentials. You can create an Anonymous user so that anyone can log in as Anonymous without having to provide a password. Anonymous connections are usually customized by the OpenSSO Enterprise administrator so that Anonymous users have limited access to the server.
Certificate	Enables a user to log in through a personal digital certificate (PDC). The user is granted or denied access to a resource based on whether or not the certificate is valid. The module can optionally require the use of the Online Certificate Status Protocol (OCSP) to determine the state of a certificate.
Data Store	Enables authentication against one or more configuration data stores within a realm.

TABLE 8–1 Authentication Service Modules *(Continued)*

Authentication Module Name	Description
Federation	Used by the service provider during federation (using SAML v1.x, SAML v2, WS-Federation, Liberty ID-FF) to create a session after validating the assertion. This authentication module can not be invoked like the other modules as it is invoked directly by the `SAMLAwareServlet`.
HTTP Basic	Enables authentication to occur with no data encryption. Credentials are validated internally using either the LDAP or Data Store authentication module.
Java Database Connectivity (JDBC)	Enables authentication through any Structured Query Language (SQL) databases that provide JDBC-enabled drivers. The SQL database connects either directly through a JDBC driver or through a JNDI connection pool.
LDAP	Enables authentication using LDAP bind, a directory server operation which associates a user identifier and password with a particular LDAP entry. You can define multiple LDAP authentication configurations for a realm.
Membership	Enables user to self-register a user entry. The user creates an account, personalizes it, and accesses it as a registered user without the help of an administrator. Implemented similarly to personalized sites such as `my.site.com`or `mysun.sun.com`.
MSISDN	The Mobile Station Integrated Services Digital Network (MSISDN) authentication module enables authentication using a mobile subscriber ISDN associated with a device such as a cellular telephone. It is a non-interactive module. The module retrieves the subscriber ISDN and validates it against the user repository to find a user that matches the number.
RADIUS	Uses an external Remote Authentication Dial-In User Service (RADIUS) server to verify identities.
Security Assertion Markup Language (SAML)	Receives and validates SAML assertions on a target server by using either a web artifact or a POST response.
SafeWord®	Uses Secure Computing's SafeWord PremierAccess™ server software and SafeWord tokens to verify identities.
SecurID™	Uses RSA ACE/Server software and RSA SecurID authenticators to verify identities.
UNIX®	Solaris and Linux modules use a user's UNIX identification and password to verify identities.

TABLE 8–1 Authentication Service Modules *(Continued)*

Authentication Module Name	Description
Windows Desktop Single Sign-On (SSO)	Allows a user who has already authenticated with a key distribution center to be authenticated by OpenSSO Enterprise without having to provide the login information again. Leverages Kerberos authentication and is supported wherever Kerberos is supported (including Windows, Solaris, Linux, and Macintosh).
Windows NT	Uses a Microsoft Windows NT™ server to verify identities.

You can use the OpenSSO Enterprise console to enable and configure the authentication modules. You can also create and configure multiple instances of a particular authentication module. (An *authentication module instance* is a child entity that extends the schema of a parent authentication module and adds its own subschema.) Finally, you can write your own custom authentication module (or plug-in) to connect to the OpenSSO Enterprise authentication framework. See Chapter 4, "Managing Authentication," in *Sun Federated Access Manager 8.0 Administration Guide* for detailed information about enabling and configuring default authentication modules and authentication module instances. See Chapter 2, "Using the Authentication Interfaces," in *Sun Federated Access Manager 8.0 Developer's Guide* for more information about writing custom authentication modules.

Authentication Types

After granting or denying access to a resource, OpenSSO Enterprise checks for information about where to redirect the user. A specific order of precedence is used when checking for this information. The order is based on whether the user was granted or denied access to the protected resource, and on the type of authentication specified. When you install OpenSSO Enterprise, a number of authentication types are automatically configured.

Realm-based Authentication.

User authenticates to a configured realm or sub-realm.

Note – This authentication type is equivalent to organization—based authentication. The query parameters `org` and `realm` would both lead to realm-based authentication in realm mode, and organization-based authentication in legacy mode.

Role-based Authentication.

User authenticates to a configured role within a realm or sub-realm. The user must possess the

role. A *static role* is possessed when an attribute is assigned to a specific user or container. A *filtered role* is dynamically generated based on an attribute contained in the user's or container's entry. For example, all users that contain a value for the `employee` attribute can be included in a role named *employees* when the filtered role is created.

Note – Role based authentication is only supported for use with the AM SDK data store schema plug-in. This data store would come from an existing Sun Java System Access Manager 7.x installation or would have been manually created. If a user installs OpenSSO Enterprise with any other user datastore, role-based authentication will not be supported.

Service-based Authentication.	User authenticates to a specific service or application registered to a realm or sub-realm.
User-based Authentication.	User authenticates using an authentication process configured specifically for him or her.
Authentication Level-based Authentication	An administrator specifies the security level of the authentication modules by defining each with an *authentication level*. Successful authentication to a higher authentication level defines a higher level of trust for the user. If a user attempts to access a service, the service can determine if the user is allowed access by checking the authentication level in the user's session data. If the authentication level is not high enough, the service redirects the user to go through an authentication process with a set authentication level.
Module-based Authentication.	Allows a user to specify the module to which they will authenticate.
Organization-based Authentication.	User authenticates to an organization or sub-organization.

Note – This authentication type is equivalent to realm-based authentication. The query parameters `org` and `realm` would both lead to realm-based authentication in realm mode, and organization-based authentication in legacy mode.

For more information, see "Authentication Types" in *Sun Federated Access Manager 8.0 Administration Guide*.

Configuring for Authentication

The authentication framework includes the following places where you can configure for authentication:

- "Core Authentication Module and Realm Configuration" on page 120
- "Authentication Configuration Service" on page 121
- "Login URLs and Redirection URLs" on page 121

Explanations of the authentication attributes can be found in the Online Help and the Part II, "Configuration Attribute Reference," in *Sun Federated Access Manager Administration Reference*.

Core Authentication Module and Realm Configuration

The Core Authentication Module contains general authentication properties that can be defined globally using the OpenSSO Enterprise console (under the Configuration tab) or more specifically for each configured realm (under the Access Control tab). Core authentication properties are added and enabled for the top-level realm during installation. As new realms are configured under the top-level realm, these properties (and the values defined globally for them) are dynamically added to each new realm when it is created. Once added, new values can be defined and configured values can be modified by the realm's administrator. The values are then used if no overriding value is defined in the specified authentication module instance or authentication chain. The default values for the Core Authentication Module are defined in the `amAuth.xml` file and stored in the configuration data store. For more information, see "General Authentication Properties" in *Sun Federated Access Manager 8.0 Administration Guide* and the Part II, "Configuration Attribute Reference," in *Sun Federated Access Manager Administration Reference*.

Authentication Configuration Service

The Authentication Configuration Service describes all the dynamic attributes for service-based authentication. This service is used for configuring roles. When you assign a service to a role, you can also assign other attributes such as a success URL or an authentication post-processing class to the role. For more information, see "Role-based Authentication" in *Sun Federated Access Manager 8.0 Administration Guide*.

Login URLs and Redirection URLs

In the last phase of the authentication process, OpenSSO Enterprise either grants or denies access to the user. If access is granted, OpenSSO Enterprise uses a login URL to display a page in the browser. If access is denied, OpenSSO Enterprise uses a redirection URL to display an alternate page in the browser. A typical alternate page contains a brief message indicating the user has been denied access.

Each authentication type (as discussed in "Authentication Types" on page 118) uses a login URL or redirection URL based on a specific order of precedence, and on whether the authentication succeeded or failed. For a detailed description of how OpenSSO Enterprise proceeds through the order of precedence, see "Authentication Types" in *Sun Federated Access Manager 8.0 Administration Guide*.

Authentication Graphical User Interfaces

The OpenSSO Enterprise Authentication Service has two separate graphical user interfaces that can be used. The following sections contain information on them.

Authentication Service User Interface

The Authentication Service implements a user interface that is separate from the OpenSSO Enterprise administration console. The Authentication Service user interface provides a dynamic and customizable means for gathering authentication credentials. When a user requests access to a protected resource, the Authentication Service presents a web-based login page and prompts the user for the appropriate credentials based on the configured authentication module or chain. Once the credentials have been passed back to OpenSSO Enterprise and authentication is deemed successful, the user may gain access to the protected resource if authorized to do so. The Authentication Service user interface can be used for the following:

- Administrators can access the administration portion of the OpenSSO Enterprise console to manage their realm's identity data.
- Users can access their own profiles to modify personal data.
- A user can access a resource defined as a redirection URL parameter appended to the login URL.
- A user can access the resource protected by a policy agent.

Below is a screen capture of the default Authentication Service user interface.

FIGURE 8–2 Authentication Service User Interface

OpenSSO Enterprise provides customization support for the Authentication Service user interface. You can customize JavaServer Pages™ (JSP™) and the file directory level by

organization, service, locale, or client type. See Chapter 16, "Customizing the Authentication User Interface," in *Sun Federated Access Manager 8.0 Developer's Guide* for more information.

Distributed Authentication User Interface

OpenSSO Enterprise also provides a remote authentication user interface component to enable secure, distributed authentication across two firewalls. A web browser communicates an HTTP request to the remote authentication user interface which, in turn, presents the appropriate module login page to the user. The web browser then sends the user login information through a firewall to the remote authentication user interface which, in turn, communicates through the second firewall with OpenSSO Enterprise. The Distributed Authentication User Interface enables a policy agent or an application that is deployed in a non-secured area to communicate with the OpenSSO Enterprise Authentication Service installed in a secured area of the deployment. Figure 8–3 illustrates this scenario.

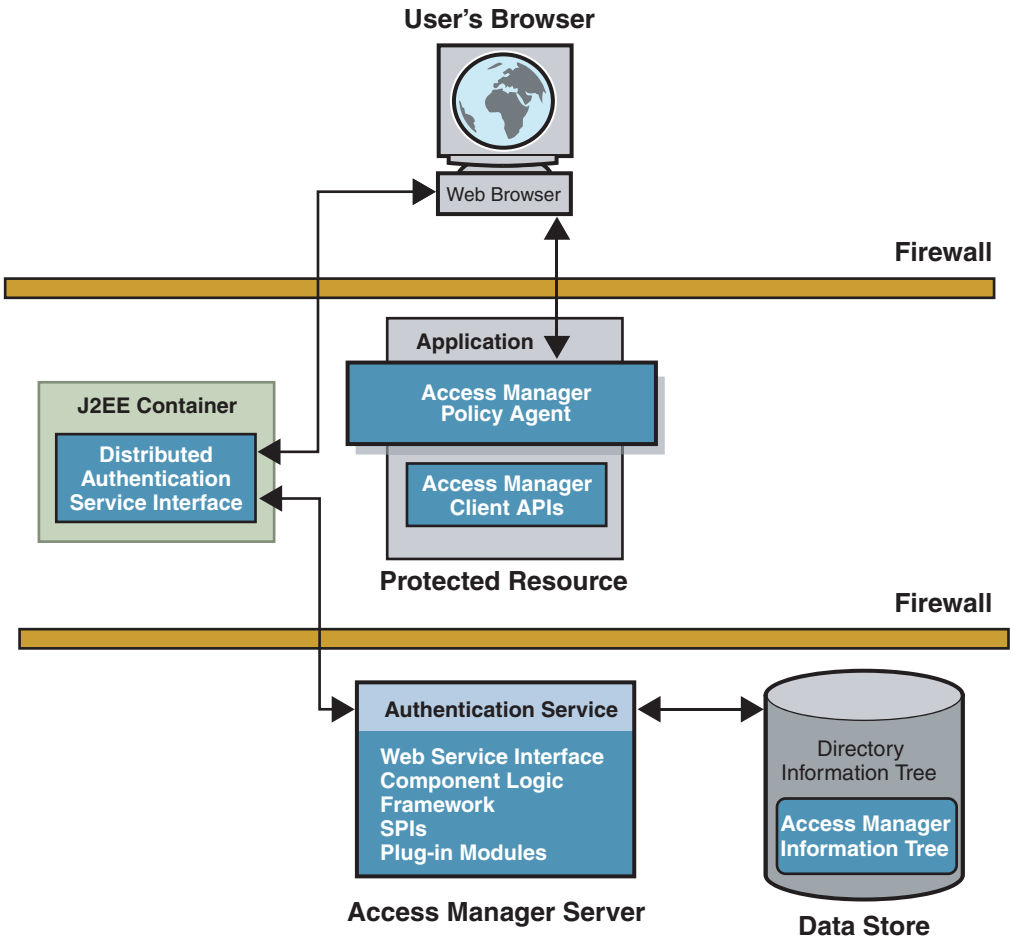


FIGURE 8–3 Distributed Authentication Process

The Distributed Authentication User Interface uses a JATO presentation framework and is customizable. (See screen capture in "Authentication Service User Interface" on page 121.) You can install the Distributed Authentication User Interface on any servlet-compliant web container within the non-secure layer of a OpenSSO Enterprise deployment. The remote component then works with the Authentication client APIs and authentication utility classes to authenticate web users. For a more detailed process, see "User Authentication" on page 93. For detailed installation and configuration instructions, see Chapter 6, "Deploying a Distributed Authentication UI Server," in *Sun OpenSSO Enterprise 8.0 Installation and Configuration Guide*.

Authentication Service Programming Interfaces

OpenSSO Enterprise provides both Java APIs and C APIs for writing authentication clients that remote applications can use to gain access to the Authenticate Service. Communication between the APIs and the Authentication Service occurs by sending XML messages over HTTP(S). The Java and C APIs support all authentication types supported by the browser-based user interface. Clients other than Java and C clients can use the XML/HTTP interface directly to initiate an authentication request. Additionally, you can add custom authentication modules to OpenSSO Enterprise by using the service provider interface (SPI) package, `com.iplanet.authentication.spi`. This SPI implements the JAAS `LoginModule`, and provides additional methods to access the Authentication Service and module configuration properties files. Because of this architecture, any custom JAAS authentication module will work within the Authentication Service. For more information, see Chapter 2, "Using the Authentication Interfaces," in *Sun Federated Access Manager 8.0 Developer's Guide* and *Federated Access Manager 8.0 Java API Reference*.

OpenSSO Enterprise also provides a Client SDK that can implement authentication logic on a remote web server or application server. For information, see Chapter 1, "Enhancing Remote Applications Using the Client Software Development Kit," in *Sun Federated Access Manager 8.0 Developer's Guide*.

◆ ◆ ◆ CHAPTER 9

Authorization and the Policy Service

The Sun OpenSSO Enterprise Policy Service determines if a user has been given permission by a recognized authority to access a protected resource. The process is referred to as *authorization*. This chapter describes how the various parts of the Policy Service work together to perform authorization. Topics covered include:

Authorization and Policy Service Overview

A *policy* is a rule that defines who is authorized to access a resource. A single policy can define authorization with either binary or non-binary decisions. (A binary decision is yes/no, true/false or allow/deny. A non-binary decision represents the value of an attribute; for example, a mail service might include a `mailboxQuota` attribute with a maximum storage value set for each user.) In general, the Policy Service allows administrators to configure, modify, and delete policies. The configured policies are then added to a *realm* and applied against the subjects in the realm. The Policy Service can be accessed using the Policy Service API: a privileged user can define access control policies using the administration API while a protected application or policy agent can obtain policy decisions using the evaluation API. The Policy Service relies on:

- A Policy Administration Point (PAP) implements the functionality to define policies. The Policy Service is the PAP.
- A Policy Enforcement Point (PEP) to protect an enterprise's resources by enforcing access control. The PEP uses the policy component of the Client SDK to retrieve policy decisions. The policy agent is the PEP.

- A Policy Decision Point (PDP) to evaluate policy and make an access determination. The Policy Service is the PDP.
- A data store in which configured policies are stored and from which they are retrieved. The Configuration Data Store is the data store.

Access to a resource is always preceded by a basic user session in which the requestor is authenticated, a session is created by the Authentication Service, and the session token identifier is validated. (See Chapter 7, "Models of the User Session and Single Sign-On Processes.") The policy agent protecting the resource then provides the session token identifier, resource name, desired action, and additional context parameters to the Policy Service which uses configured policies to determine if the user has been given permission to access the protected resource by a recognized authority. When the policy agent gets the decision from the Policy Service, it allows or denies access to the user, enforcing the policy decision provided by Policy Service. This whole process is referred to as *authorization*. The Policy Service is defined by the `amPolicy.xml` and, generally speaking:

- Provides a means for defining and managing access policies.
- Provides a means for defining custom policy plug-ins by providing names and class locations.
- Evaluates access policies.
- Acts as a PDP to deliver the result of a policy evaluation.
- Supports the delegation of policy management.
- Provides an SPI for extensibility.
- Provides access from remote clients using the Client SDK.
- Caches and reuses policy decisions, where applicable, to improve performance.
- Allows periodic polling of the Policy Service by a client to update locally cached policy decisions.
- Dynamically recognizes changes to policies and provides policy decisions that reflect them.

Note – The Policy Configuration Service provides a means to specify how policies are defined and evaluated. It enables you to specify, for example, which directory to use for subject lookup, the directory password, which search filters to use, and which subjects, conditions, and response providers to use. This configuration can be done within a realm or a subrealm and is accessible using the OpenSSO Enterprise console.

See Chapter 5, "Managing Policies," in *Sun Federated Access Manager 8.0 Administration Guide* and Chapter 3, "Enforcing Authorization with the Policy Service," in *Sun Federated Access Manager 8.0 Developer's Guide* for more information.

Policy Types

The Policy Service authorizes access to a user based on policies created and stored in the OpenSSO Enterprise configuration data store. The following sections contain information on the two types of policies you can create.

- "Normal Policy" on page 129
- "Referral Policy" on page 132

For more information, see Chapter 5, "Managing Policies," in *Sun Federated Access Manager 8.0 Administration Guide*.

Normal Policy

A *normal policy* specifies a protected resource and who is allowed to access it. The protected resource can be anything hosted on a protected server. Examples of protected resources are applications, document files, images, or the server itself. A normal policy consists of *rules*, *subjects*, *conditions*, and *response providers*. The following sections contain information on these elements.

- "Rules" on page 129
- "Subjects" on page 129
- "Conditions" on page 130
- "Response Providers" on page 131

Rules

A *rule* defines the policy itself by specifying a resource, one or more sets of an action, and values for each action.

- A *resource* defines the specific object that is being protected. Examples of protected objects are an HTML page on a web site, or a user's salary information accessed using a human resources service.
- An *action* is the name of an operation that can be performed on the resource. Examples of web page actions are POST and GET. An allowable action for a human resources service might be `canChangeHomeTelephone`.
- A *value* defines the permission for the action. Examples are `allow` and `deny`.

Subjects

A *subject* specifies the user or collection of users that the policy affects. The following list of subjects can be assigned to policies.

Access Manager Identity Subjects	The identities you create and manage under the Subjects tab in a configured realm can be added as a value of the subject.

Authenticated Users	Any user with a valid session (even if they have authenticated to a realm that is different from the realm in which the policy is defined) is a member of this subject. This is useful if the resource owner would like to allow access to users from other organizations. To restrict a resource's access to members of a specific organization, use the Organization subject.
Web Services Clients	This implies that a web service client (WSC) identified by a session token identifier is a member of this subject — as long as the distinguished name (DN) of any principal contained in the session token identifier matches any selected value of this subject.

The following list of subjects can only be specified after they are selected using the Policy Configuration Service of the appropriate realm.

OpenSSO Enterprise Roles	Any member of a OpenSSO Enterprise role is a member of this subject. A OpenSSO Enterprise role is created using OpenSSO Enterprise running in legacy mode. These roles have object classes mandated by OpenSSO Enterprise and can only be accessed through the hosting OpenSSO Enterprise Policy Service.
	Note – This subject can be used when connected to an AMSDK data store.
LDAP Groups	Any member of an LDAP group can be added as a value of this subject.
LDAP Roles	Any LDAP role can be added as a value of this subject. An LDAP Role is any role definition that uses the Sun Java System Directory Server role capability. These roles have object classes mandated by Directory Server role definition. The LDAP Role Search filter can be modified in the Policy Configuration Service to narrow the scope and improve performance.
LDAP Users	Any LDAP user can be added as a value of this subject.
Organization	Any member of a realm is a member of this subject.

Conditions

A *condition* specifies additional constraints that must be satisfied for a policy be applicable. For example, you can define a condition to limit a user's network access to a specific time period.

The condition might state that the subject can access the network only between 7:00 in the morning and 10:00 at night. OpenSSO Enterprise allows for the following list of conditions.

Active Session Time	Sets a condition based on constraints configured for user session time such as maximum session time.
Authentication Chain	The policy is applicable if the user has successfully authenticated to the authentication chain in the specified realm. If the realm is not specified, authentication to any realm at the authentication chain will satisfy the condition.
Authentication Level	The Authentication Level attribute indicates the level of trust for authentication. The policy is applicable if the user's authentication level is greater than or equal to the Authentication Level set in the condition, or if the user's authentication level is less than or equal to the Authentication Level set in the condition, depending on the configuration.
Authentication Module Instance	The policy applies if the user has successfully authenticated to the authentication module in the specified realm. If the realm is not specified, authentication to any realm at the authentication module will satisfy the condition.
IP Address/DNS Names	Sets a condition based on a range of IP Addresses, or a DNS name.
Current Session Properties	Decides whether a policy is applicable to the request based on values set in the user's Access Manager session.
LDAP Filter Condition	The policy is applicable when the defined LDAP filter locates the user entry in the LDAP directory that was specified in the Policy Configuration service.
Realm Authentication	The policy applies if the user has authenticated to the specified realm.
Time	Sets the condition based on time constraints (time, day, date, time zone).

Response Providers

Response providers are plug-ins that provide policy response attributes. Policy response attributes typically provide values for attributes in the user profile. The attributes are sent with policy decisions to the PEP which, in turn, passes them in headers to an application. The application typically uses these attributes for customizing pages such as a portal page. OpenSSO

Enterprise includes one implementation of the `com.sun.identity.policy.interfaces.ResponseProvider` class, the `IDResponseProvider`. See Chapter 3, "Enforcing Authorization with the Policy Service," in *Sun Federated Access Manager 8.0 Developer's Guide* for more information.

Referral Policy

A user with the Top—level Realm Administrator or Policy Administrator roles can create policy. (A Realm Administrator or Policy Administrator configured for a specific realm have permission to create policies only for resources delegated to that realm.) A *referral policy* enables either administrator to delegate policy configuration tasks. A referral policy delegates both policy creation and policy evaluation, and consists of one or more rules and one or more referrals.

- A *rule* defines the resource of which policy creation or evaluation is being referred.
- A *referral* defines the identity to which the policy creation or evaluation is being referred.

Referral policies delegate policy management privileges to another entity such as a peer realm, a subrealm, or even a third-party product. (You can implement custom referrals by using the Policy APIs.) For example, assume a top-level realm exists named `ISP`. It contains two subrealms: `company1` and `company2`. The Top-Level Realm Administrator for `ISP` can delegate policy management privileges so that a Realm Administrator in `company1` can create and manage policies only within the`company1` realm, and a Realm Administrator in `company2` can create and manage policies only within the `company2` realm. To do this, the Top-Level Realm Administrator creates two referral policies, defining the appropriate realm in the rule and the appropriate administrator in the referral. See "Creating Policies" in *Sun Federated Access Manager 8.0 Administration Guide* for more information.

Realms and Access Control

When a user logs into an application, OpenSSO Enterprise plug-ins retrieve all user information, authentication properties, and authorization policies that the OpenSSO Enterprise framework needs to form a temporary, virtual user identity. The Authentication Service and the Policy Service use this virtual user identity to authenticate the user and enforce the authorization policies, respectively. All user information, authentication properties, and authorization policies is contained within a *realm*. You create a realm when you want to apply policies to a group of related subjects, services or servers. The Policy Configuration Service within the realm provides a means to specify how policies are defined and evaluated. It enables you to specify, for example, which directory to use for subject lookup, the directory password, which search filters to use, and which subjects, conditions, and response providers to use. For example, you can create a realm that groups all servers and services that are accessed regularly

by employees in one region. And, within that regional grouping realm, you can group all servers and services accessed regularly by employees in a specific division such as Human Resources. A configured policy might state that all Human Resources administrators can access the URL `http://HR.example.com/HRadmins/index.html`. You might also add constraints to this policy: it is applicable only Monday through Friday from 9:00 a.m. through 5:00 p.m. Realms facilitate the delegation of policy management privileges. These configurations can be done within a realm or a sub realm and is accessible using the OpenSSO Enterprise console.

Note – Access control realms can be configured to use any supported user database.

Policy Service Programming Interfaces

OpenSSO Enterprise provides both Java API and C API for writing clients that remote applications can use to administer policies and evaluate policy decisions. They are used to add, lookup, modify or replace policies, and to evaluate policy decisions when a principal attempts an action on a protected resource. Communication between the API and the Policy Service occurs by sending XML messages over HTTP(S). Additionally, you can extend and customize the Policy Service using the SPI. The classes are used by service developers and policy administrators who need to provide additional policy features as well as support for legacy policies. For example, you can develop customized plug-ins for creating custom policy subjects, referrals, conditions, and response providers. Lastly, the Client SDK is provided to implement policy evaluation logic on a remote web server or application server. For information, see Chapter 1, "Enhancing Remote Applications Using the Client Software Development Kit," in *Sun Federated Access Manager 8.0 Developer's Guide*, Chapter 3, "Enforcing Authorization with the Policy Service," in *Sun Federated Access Manager 8.0 Developer's Guide*, the *Sun Federated Access Manager 8.0 C API Reference*, and the *Federated Access Manager 8.0 Java API Reference*.

XACML Service

eXtensible Access Control Markup Language (XACML) is a markup language that provides an XML syntax for defining policies (who can do what, where can it be done, and when), for querying whether access to a protected resource can be allowed (requests), and for receiving responses to those queries (decisions). XACML is built around the standard access control separation of the Policy Enforcement Point (PEP) and the Policy Decision Point (PDP) as discussed in "Authorization and Policy Service Overview" on page 127 except you use XACML formatted queries and responses. The XACML PEP is responsible for intercepting all access requests, collecting the appropriate information (such as who is making the request, which resource is being accessed, and what action is to be taken), and sending a request for a decision to the XACML PDP. The XACML PDP (OpenSSO Enterprise) evaluates configured policies

against the information in the decision request. It uses a Context Handler to request the appropriate policies and attributes in order to render one of the following decisions.

- Permit
- Deny
- Not Applicable (no policy created by this PDP applies to the access request)
- Indeterminate (an error occurred that prevents the PDP from knowing the correct response)

The following sections contain more information.

- "XACML in OpenSSO Enterprise" on page 134
- "XACML Programming Interfaces" on page 136

XACML in OpenSSO Enterprise

OpenSSO Enterprise implements the SAML v2 Profile of XACML version 2.0 thus supporting `XACMLAuthzDecisionQuery` and `XACMLAuthzDecisionStatement`. In a OpenSSO Enterprise XACML interaction, after receiving a request for access, the XACML PEP makes a `XACMLAuthzDecisionQuery` request and receives a `XACMLAuthzDecisionStatement` response that contains the decision. (The policies themselves are not returned.) The XACML components on the client side include Client SDK interfaces for passing XACML requests and receiving XACML responses as well as an interface to construct the communications.

Note – The framework relies internally on the Client SDK SAML v2 interfaces for communication between the PEP and PDP, and includes an implementation of the SAML v2 request handler called the `XACML2AuthzDecisionQueryHandler` that plugs into the SAML v2 Service framework.

The XACML components on the OpenSSO Enterprise side include out-of-the-box implementations of XACML mappers for subjects, resources, actions and environment. These implementations use the Policy Service to compute authorization decisions. XXXXXX illustrates how XACML and OpenSSO Enterprise interact with each other. The communications are explained more fully following the image. XXXXXX

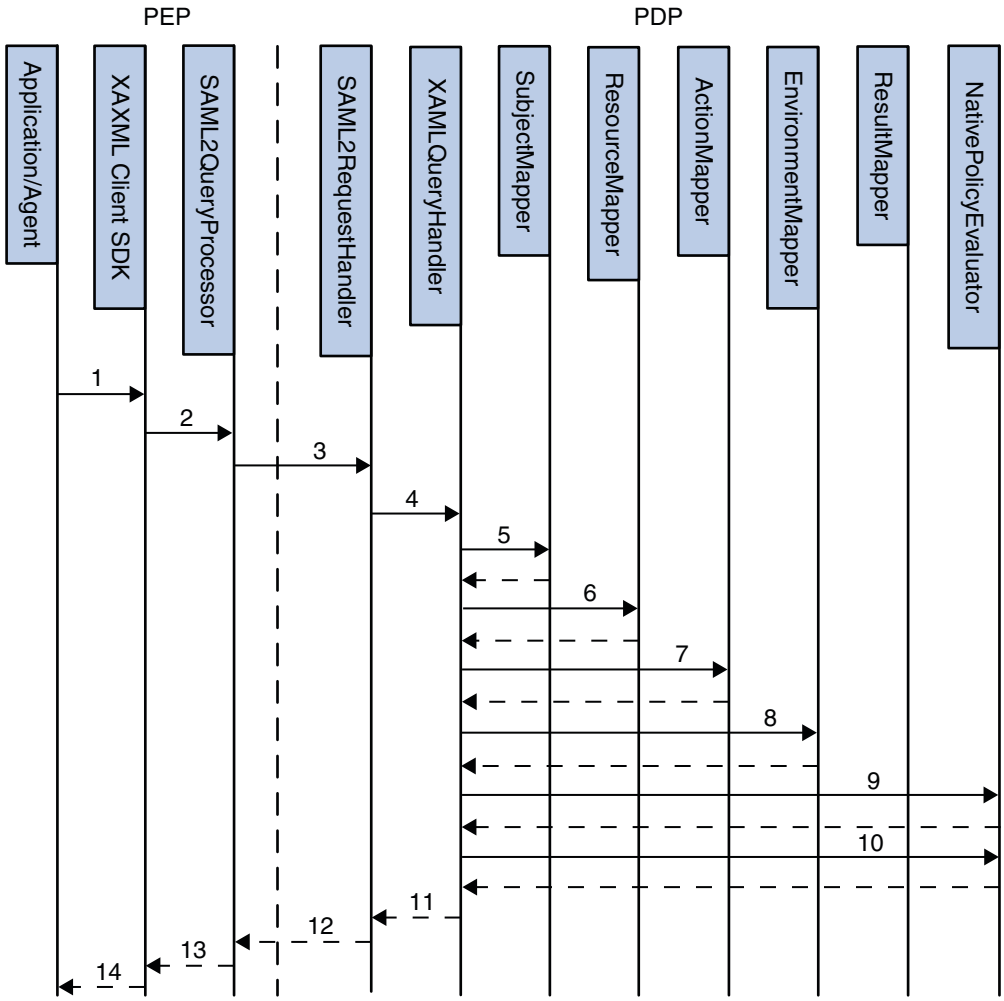


FIGURE 9–1 XACML Process Flow

1. The policy agent protecting a resource constructs a XACML access request using the Client SDK.
2. The Client SDK wraps the request in a `XACMLAuthzDecisionQuery` element and sends it to the SAML v2 query processor on the local machine (also part of the Client SDK).
3. The SAML v2 query processor consults the metadata for the PEP and the PDP, sets additional elements or attributes in the query, signs it (if necessary) and sends a SOAP request containing the query to the PDP.

4. The SAML v2 request handler on the PDP side receives the request, consults the metadata for the PEP and the PDP, verifies the trust relationships, enforces any signing or encryption requirements, verifies the signature and forwards the query to the `XACMLAuthzDecisionQueryHandler`.
5. The `XACMLAuthzDecisionQueryHandler` consults the appropriate metadata using the `entityID` values of the PEP and PDP (included in the request) to find the correct mapper implementations to use.
6. `XACMLAuthzDecisionQueryHandler` uses the Resource mapper to map the given Resource to a resource and service configured with OpenSSO Enterprise.
7. `XACMLAuthzDecisionQueryHandler` uses the Action mapper to map the given Action to an action name configured with OpenSSO Enterprise.
8. `XACMLAuthzDecisionQueryHandler` uses the Environment mapper to map the given Environment to conditions configured with OpenSSO Enterprise.
9. `XACMLAuthzDecisionQueryHandler` uses the OpenSSO Enterprise policy evaluator to get the policy decision.
10. `XACMLAuthzDecisionQueryHandler` uses the Result mapper to map the decision to an XACML Result element.

Note – OpenSSO Enterprise is not an XACML policy engine. It has no support for XACML policies themselves and thus no support for retrieving the policies, only the decision.

---

11. `XACMLAuthzDecisionQueryHandler` wraps the XACML Result in an XACML Response, the XACML Response in an `XACMLAuthzDecisionStatement`, the `XACMLAuthzDecisionStatement` in a SAML Assertion, the Assertion in a SAML Response, and hands over the SAML Response to the SAML v2 request handler.
12. The SAML v2 request handler sets additional attributes and elements (based on the SAML v2 protocol), signs it as required and returns it in a SOAP message to the PEP side.
13. The SAML v2 query processor verifies the trust relationships, the signing requirements, and the signature as necessary. It then extracts the SAML Response from the SOAP message and returns it to the XACML portion of the Client SDK.
14. The Client SDK extracts the XACML Response from the SAML v2 Response and returns it (and the decision) to the client application.

XACML Programming Interfaces

OpenSSO Enterprise provides Java API for using, and interacting with, the XACML Service. For information, see Chapter 1, "Enhancing Remote Applications Using the Client Software Development Kit," in *Sun Federated Access Manager 8.0 Developer's Guide*, Chapter 3,

"Enforcing Authorization with the Policy Service," in *Sun Federated Access Manager 8.0 Developer's Guide*, and the *Federated Access Manager 8.0 Java API Reference*.

PART III

Federation Management Using OpenSSO Enterprise

Sun OpenSSO Enterprise provides a framework for implementing a federated identity infrastructure, enabling single sign-on, provisioning users dynamically, and sharing identity attributes across security domains. The chapters in this third part of the Sun OpenSSO Enterprise Technical Overview contains information on federation management.

◆ ◆ ◆ CHAPTER 10

What is Federation?

Federation establishes a standards-based method for sharing and managing identity data and establishing single sign-on across security domains and organizations. It allows an organization to offer a variety of external services to trusted business partners as well as corporate services to internal departments and divisions. Forming trust relationships across security domains allows an organization to integrate applications offered by different departments or divisions within the enterprise as well as engage in relationships with cooperating business partners that offer complementary services. Towards this end, multiple industry standards, such as those developed by the Organization for the Advancement of Structured Information Standards (OASIS) and the Liberty Alliance Project, are supported. This chapter contains an overview of federation.

The Concept of Federation

As a concept, federation encompasses both *identity federation* and *provider federation*.

Identity Federation

In one dictionary, *identity* is defined as "a set of information by which one person is definitively distinguished." This information undoubtedly begins with the document that corroborates a person's name: a birth certificate. Over time, additional information further defines different aspects of an individual's identity. The composite of this data constitutes an identity with each

specific piece providing a distinguishing characteristic. Each of the following represents data that designates a piece of a person's identity as it relates to the enterprise for which the data was defined.

- An address
- A telephone number
- One or more diplomas
- A driver's license
- A passport
- Financial institution accounts
- Medical records
- Insurance statements
- Employment records
- Magazine subscriptions
- Utility bills

Because the Internet is now one of the primary vehicles for the types of interactions represented by identity-defining information, people are creating online identities specific to the businesses with which they are interacting. By creating a user account with an identifier and password, an email address, personal preferences (such as style of music, or opt-in/opt-out marketing decisions) and other information specific to the particular business (a bank account number or ship-to address), a user is able to distinguish their account from others who also use the enterprise's services. This distinguishing information is referred to as a *local identity* because it is specific to the *service provider* (a networked entity that provides one or more services to other entities) for which it has been defined. Sending and receiving email, checking bank balances, finalizing travel arrangements, accessing utility accounts, and shopping are just a few online services for which a user might define a local identity. If a user accesses all of these services, many different local identities have been configured. Considering the number of service providers for which a user can define a local identity, accessing each one can be a time-consuming and frustrating experiencing. In addition, although most local identities are configured independently (and fragmented across the Internet), it might be useful to connect the information. For example, a user's local identity with a bank could be securely connected to the same user's local identity with a utility company for easy, online payments. This virtual phenomenon offers an opportunity for a system in which users can *federate* these local identities. *Identity federation* allows the user to link, connect, or bind the local identities that have been created for each service provider. The linked local identities, referred to as a *federated identity*, allow the user to log in to one service provider site and click through to an affiliated service provider without having to reauthenticate or reestablish identity; in effect, single sign-on (SSO).

Provider Federation

Provider federation begins with a circle of trust. A *circle of trust* is a group of service providers who contractually agree to exchange authentication information. Each circle of trust must include at least one *identity provider*, a service provider that maintains and manages identity

data, and provides authentication services. After the business contracts and policies defining a circle of trust are in place, the specific protocols, profiles, endpoints, and security mechanisms being used by each member is collected into a metadata document that is exchanged among all other members of the circle. OpenSSO Enterprise provides the tools necessary to integrate the metadata and enable a circle of trust technologically. Authentication within this federation is honored by all membered providers.

Note – The establishment of contractual trust agreements between providers is beyond the scope of this guide. See "The Concept of Trust" on page 143 for an overview.

The Concept of Trust

Federating identities assumes existing trust relationships between participants. This trust is usually defined through business arrangements or contracts that describe the technical, operational, and legal responsibilities of each party and the consequences for not completing them. When defined, a trust relationship allows one organization to trust the user authentication and authorization decisions of another organization. This trust then enables a user to log in to one site and, if desired, access a trusted site without reauthentication.

Ensure that trust agreements are in force before configuring circles of trust with OpenSSO Enterprise and going live. The Liberty Alliance Project has created a support document for helping to establish these trust arrangements. The *Liberty Trust Model Guidelines* document is located on the Support Documents and Utility Schema Files page of the Liberty Alliance Project web site.

How Federation Works

The goal of federation is to enable individuals and service providers to protect identity data while conducting network transactions across secure domains. When organizations form a trust agreement, they agree to exchange user authentication information using specific web technologies. The trust agreement would be among multiple service providers that offer web-based services to users and, at least, one *identity provider* (a service provider that maintains and manages identity information). Once *metadata* (a particular provider's federation configuration information) is exchanged and the trust is established technologically, single sign-on can be enabled between all the included providers, and users may opt to federate their multiple identities (depending on the protocol being used). In OpenSSO Enterprise, the trust agreement is virtually configured as a *circle of trust* using the console or command line interface. A circle of trust contains *providers* (service providers or identity providers) that are grouped together for the purpose of offering identity federation. *Identity federation* occurs when a user chooses to unite distinct service provider and identity provider accounts while retaining the individual account information with each provider. The user establishes a link

that allows the exchange of authentication information between provider accounts. Users can choose to federate any or all identities they might have. After identity federation, when a user successfully authenticates to one of the service providers, access to any of the federated accounts within the circle of trust is allowed *without having to reauthenticate*. The following figure shows the subjects involved in federation.

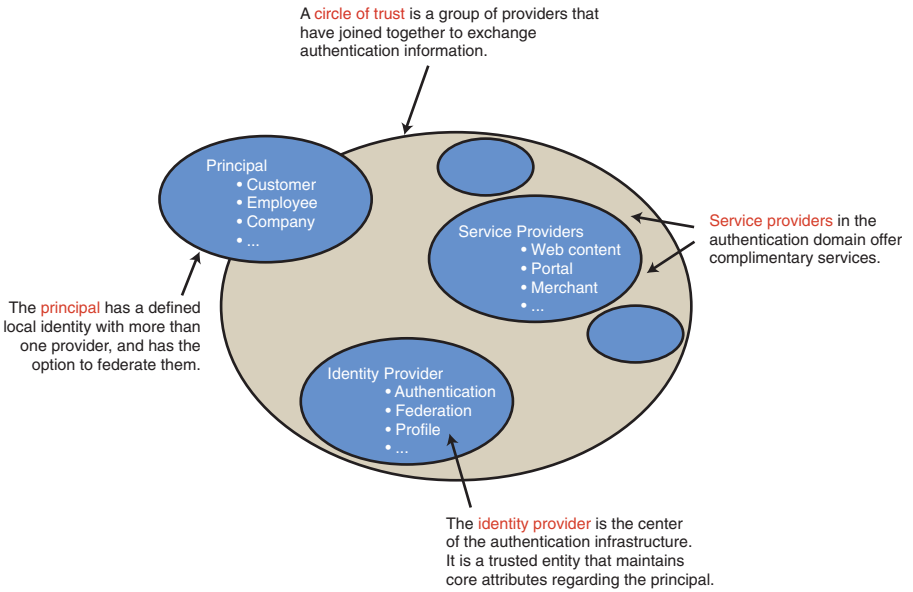


FIGURE 10–1 Subjects Involved in an Identity Federation

- A *principal* can have a defined local identity with more than one provider, and it has the option to federate the local identities. The principal might be an individual user, a group of individuals, a corporation, or a component of the Liberty architecture.
- A *service provider* is a commercial or not-for-profit organization that offers a web-based service such as a news portal, a financial repository, or retail outlet.
- An *identity provider* is a service provider that stores identity profiles and offers incentives to other service providers for the prerogative of federating their user identities. Identity providers might also offer services above and beyond those related to identity profile storage.
- To support identity federation, all service providers and identity providers must join together into a *circle of trust*. A circle of trust must contain at least one identity provider and at least one service provider. (One organization may be both an identity provider and a service provider.) Providers in a circle of trust must first write trust agreements to define their relationships. A *trust agreement* is a contract between organizations that defines how the circle will work. For more information, see "The Concept of Trust" on page 143.

A travel portal is a good example of a circle of trust. Typically, a travel portal is a web site designed to help you access various travel-related services from one location. The travel portal forms a partnership with each service provider displayed on its web site. (This might include hotels, airlines, and car rental agencies.) The user registers with the travel portal which, in effect, is the identity provider for the circle of trust. After logging in, the user might click through to an airline service provider to look for a flight. After booking a flight, the user might click through to an accommodations service provider to look for a hotel. Because of the trust agreements previously established, the travel portal shares authentication information with the airline service provider, and the airline service provider with the accommodations service provider. The user moves from the hotel reservations web site to the airline reservations web site without having to reauthenticate. All of this is transparent to the user who must, depending on the underlying federation protocol, choose to federate any or all local identities. The following figure illustrates the travel portal example.

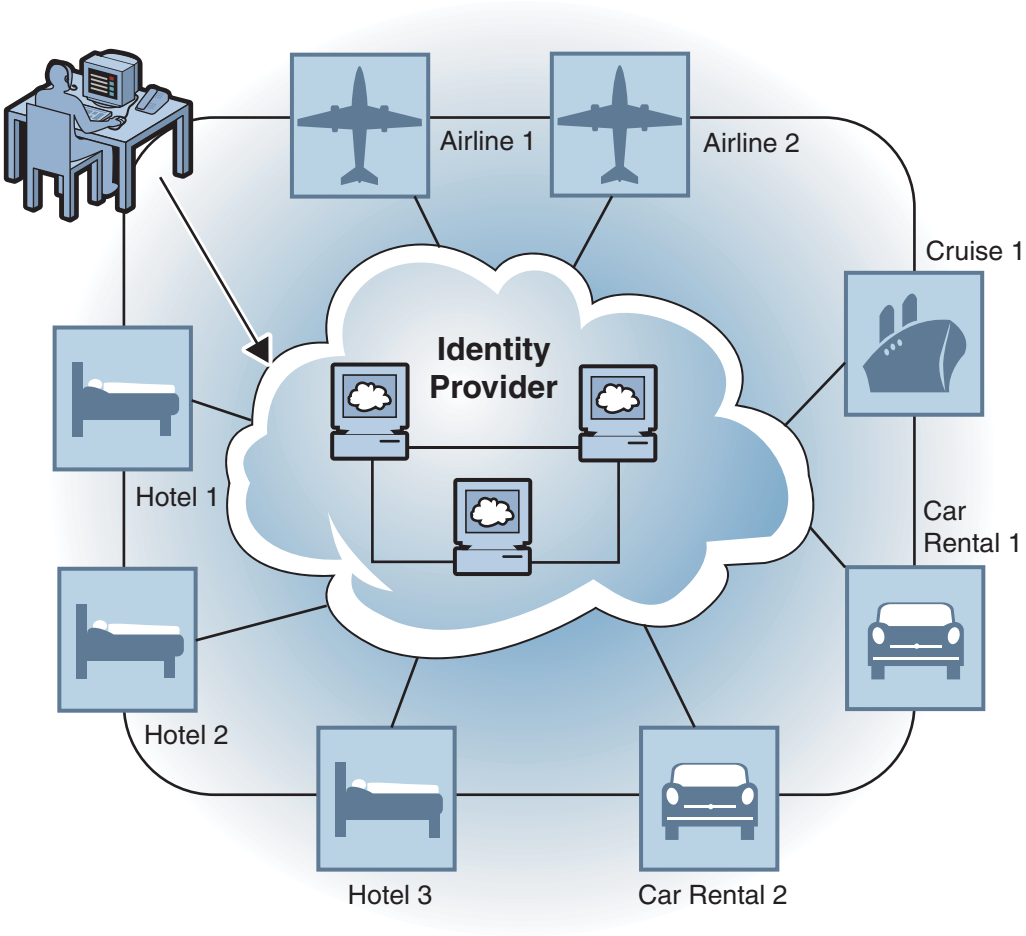


FIGURE 10–2 Federation Within a Travel Portal

◆ ◆ ◆ CHAPTER 11

Federation Management with OpenSSO Enterprise

Sun OpenSSO Enterprise provides a pluggable framework for implementing federated identity infrastructures. The Federation framework places no restrictions on the use of network technologies, computer hardware, operating systems, programming languages or other hardware or software entities. It is based on, and conforms to, open industry standards to achieve interoperability among different vendors on heterogeneous systems, and provides the facility to log identity interactions and erroneous conditions. The following sections contain information about the federation framework.

Key Federation Management Features

OpenSSO Enterprise creates a comprehensive security and identity management framework optimized to work with, and extend, an identity provider's existing security infrastructure. The following list describes some key features:

- Exchange of credentials and security tokens across circle of trust partners for purposes of authentication and single sign-on.
- Automatic federation of user accounts across multiple security domains.
- Session management across authentication domains to determine when user interactions must be terminated (single logout).
- Import or export the data required to establish basic federated communication between providers.
- Manages and links providers that are available to participate in a circle of trust.
- Searches for available end points and identifies each provider's federation capabilities.
- Exchanges SAML security assertions among providers in a circle of trust.
- Data management choices include an LDAPv3 directory (OpenDS, Sun Java System Directory Server or Microsoft Active Directory).

- Included service provider interfaces (SPIs) to allow customized logic during the federation process.
- Support for bulk federation and auto federation.
- Support for The Fedlet, a web archive (WAR) of data that can be embedded into a service provider application.
- Support for Virtual Federation.
- Support for multiple federation protocols in one circle of trust.

The following sections contain additional information on the final three features listed.

- "The Fedlet" on page 148
- "Virtual Federation" on page 148
- "Multi-Federation Protocol Hub" on page 151

The Fedlet

Fedlet is the name given to `fedlet.war`. The WAR is a very small archive of a few JARs, properties, and metadata (all stored in flat files) that can be embedded into a service provider's Java EE web application to allow for SSO between an identity provider instance of OpenSSO Enterprise and the service provider application - WITHOUT installing OpenSSO Enterprise on the service provider side. The service provider simply downloads the Fedlet, modifies their application to include the Fedlet JARs and, re-archives and redeploys the modified application. The service provider is now able to accept an HTTP POST (that contains a SAML v2 assertion) from the identity provider and retrieve included user attributes to accomplish SSO. (Currently, the Fedlet only supports the HTTP POST Profile.). The Fedlet can communicate with multiple identity providers, using a Discovery Service to find the preferred identity provider. XXXXXX LINK

Virtual Federation

Virtual Federation provides a mechanism for one application to communicate identity information to a second application in a different domain. In essence, it provides a secure gateway that enables legacy applications to communicate authentication attributes without having to deal specifically with federation protocols and processing. Virtual Federation allows:

- Identity provider applications to push user authentication, profile and transaction information to a local instance of OpenSSO Enterprise which then passes the data to a remote instance of OpenSSO Enterprise at the service provider using federation protocols.
- Service provider applications to consume the received information.

Virtual Federation uses SAML v2 to transfer identity data between the communicating entities. The Client SDK (which contains Java and .NET interfaces) runs independent of OpenSSO Enterprise and enables existing applications to handle the SAML v2 interactions. The following diagram illustrates this scenario.

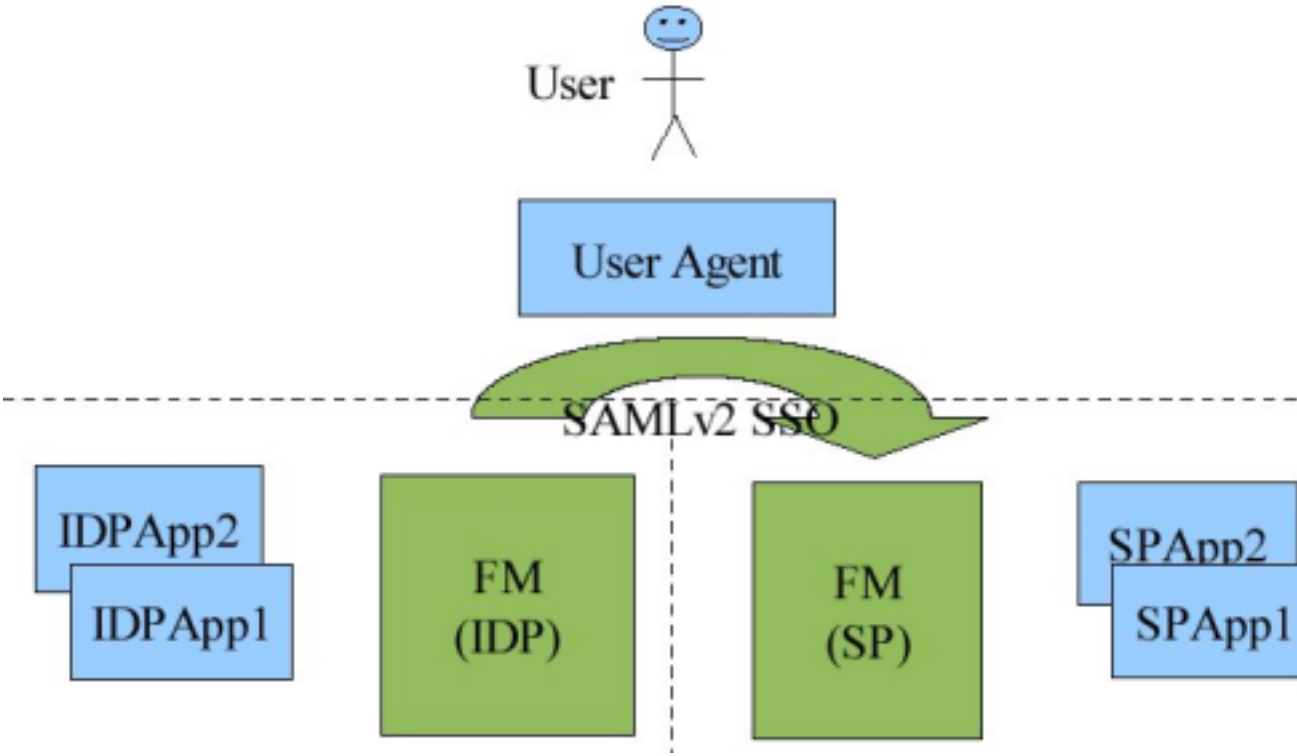


FIGURE 11–1 Virtual Federation Architecture

The following use cases are applicable to Virtual Federation:

- "Authentication at Identity Provider" on page 149
- "Virtual Federation at Identity Provider" on page 150
- "Virtual Federation at Service Provider" on page 150
- "Global Single Logout" on page 150

Authentication at Identity Provider

When a user is already authenticated within an enterprise, the legacy identity provider application sends a secure HTTP GET/POST message to OpenSSO Enterprise asserting the identity of the user. OpenSSO Enterprise verifies the authenticity of the message and establishes

a session for the authenticated user. You can use Virtual Federation to transfer the user's authentication information to the local instance of OpenSSO Enterprise in order to create the session.

Virtual Federation at Identity Provider

When a user is already authenticated by, and attempts access to, a legacy identity provider application, the legacy application sends a secure HTTP POST message to the local instance of OpenSSO Enterprise asserting the user's identity, and containing a set of attribute/value pairs related to the user (for example, data from the persistent store representing certain transactional states in the application). OpenSSO Enterprise verifies the authenticity of the message, establishes a session for the authenticated user, and populates the session with the user attributes.

Virtual Federation at Service Provider

When a user is already authenticated by the instance of OpenSSO Enterprise at the identity provider and invokes an identity provider application that calls for redirection to a service provider, the identity provider invokes one of the previous use cases and encodes a SAML v2 SSO URL as a part of the request. The identity provider instance of OpenSSO Enterprise then initiates SAML v2 SSO with the instance of OpenSSO Enterprise at the service provider. The service provider's instance of OpenSSO Enterprise then verifies the SAML v2 assertion and included attributes, and redirects to the service provider application, securely transferring the user attributes via a secure HTTP POST message. The service provider application consumes the attributes, establishes a session, and offers the service to the user.

Global Single Logout

When a user is already authenticated and has established, for example, SSO with the instance of OpenSSO Enterprise at the service provider, the user might click on a Global Logout link. The identity provider will then invalidate its local session (if created) and trigger SAML v2 single log out by invoking a provided OpenSSO Enterprise URL. The OpenSSO Enterprise identity provider executes the SAML v2 single log out, terminating the session on both provider instances of OpenSSO Enterprise.

Note – An identity provider side application can initiate single logout by sending `sun.cmd=logout` attributes via a Virtual Federation interaction to a local instance of OpenSSO Enterprise acting as the identity provider. In turn, this instance will execute SAML v2 single logout based on the current session.

For more information, see XXXXXX LINK.

Multi-Federation Protocol Hub

OpenSSO Enterprise allows a configured circle of trust to contain entities speaking different federation protocols thus supporting cross protocol single sign-on and logout among hosted identity providers in the same circle of trust. For example, you can create a circle of trust containing one identity provider instance that communicates with multiple federation protocol and three service provider instances that speak, respectively, Liberty ID-FF, SAML v2 and WS-Federation. Figure 11–2 illustrates the process of multi-federation protocol single sign-on and single logout.

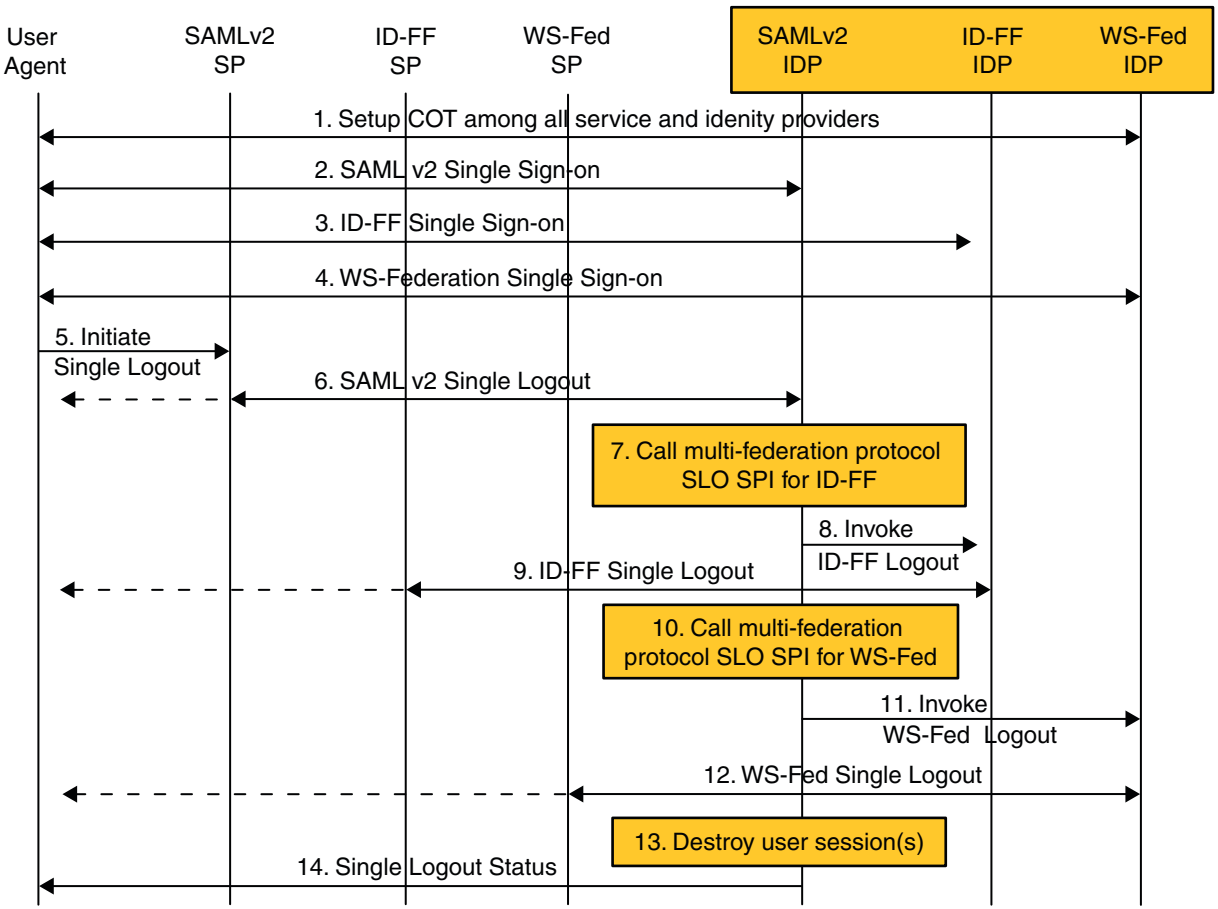


FIGURE 11–2 Multi-Federation Protocol Single Sign-on and Single Logout

For more information, see XXXXXX LINK.

The Federation Framework Architecture

OpenSSO Enterprise consists of web-based services [using SOAP, XML over HTTP(S) or HTML over HTTP(S)], and Java—based application provider interfaces (APIs) and service provider interfaces (SPIs). The figure below illustrates this architecture. Additionally, the figure shows an agent embedded into a web container. This agent enables the service provider applications to participate in the SAML or Liberty-based protocols. The darker boxes are components provided by OpenSSO Enterprise.

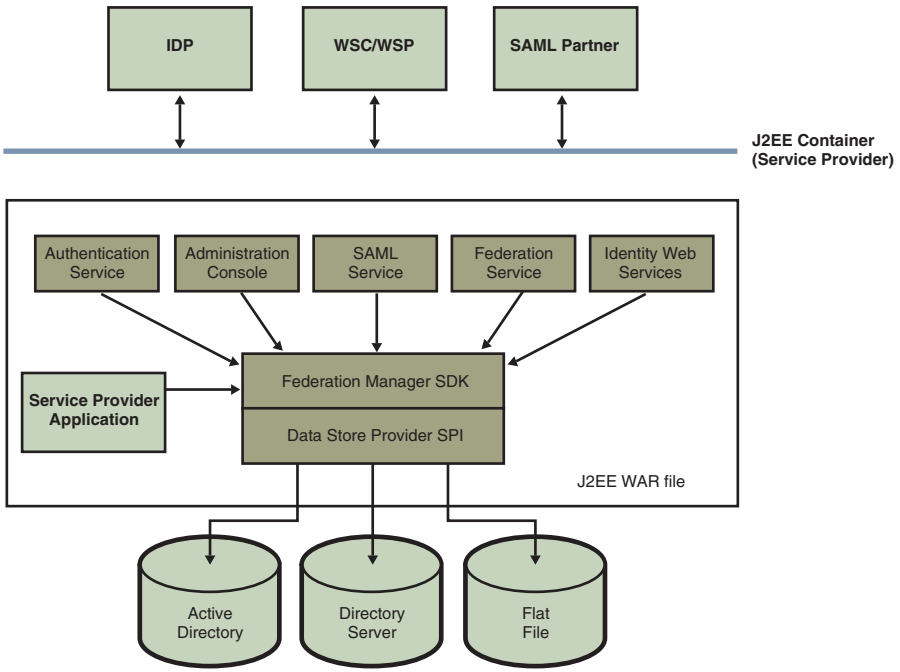


FIGURE 11–3 Federation Framework Architecture

The components include:

SAML Service	Provides SAML related services (versions 1.x and 2.0) including artifact and POST profile support, and assertion query support.
Federation Services	Provides services based on federation specifications. Features include federation and single sign-on, single logout, federation termination, name registration, and support for the Common Domain. Implemented web services include a SOAP binding service, a discovery service, a personal profile service, and an authentication service.

Authentication	OpenSSO Enterprise provides a JAAS-based authentication framework.
Session	OpenSSO Enterprise provides session management for service provider applications.
Logging	OpenSSO Enterprise provides a logging service. It also provides activity logs for auditing. Audit logs can be stored in flat files or JDBC-compliant databases.
APIs	Includes a set of APIs for interaction between the SSO, logging, SAML, federation, and authentication components. Also included are APIs to build web services for clients and providers.
SPIs	Includes a set of Service Provider Interfaces (SPIs) into which applications can insert their custom logic. For instance, there is an SPI to do post federation processing, and an SPI for post processing after a successful single logout.

◆ ◆ ◆ CHAPTER 12

Choosing a Federation Option

Sun OpenSSO Enterprise supports multiple federation standards, such as those developed by the Organization for the Advancement of Structured Information Standards (OASIS) and the Liberty Alliance Project. This chapter contains information on these federation options.

Federation Options

Federation is used to solve the problem of cooperation across heterogeneous, autonomous environments. In the beginning, federation meant using the Liberty Alliance Project Identity Federation Framework (Liberty ID-FF). Since then, other specifications have been developed with which federation can be accomplished including the Security Assertion Markup Language (SAML) and WS-Federation. To get started, use SAML v2 whenever possible as it supersedes both the Liberty ID-FF and SAML v1.x specifications.

Note – The basic difference between the proprietary CDSSO (as described in Part II) and SAML v2 is that CDSSO uses a single authentication authority, a mechanism to move a cookie between multiple DNS domains. SAML v2, on the other hand, gives you the option of using multiple authentication authorities, with one authority asserting the identity of the user to the other.

SAML v1.x was designed to address the issue of cross-domain single sign-on. It does not solve issues such as privacy, single logout, and federation termination. The Liberty Alliance Project was formed to develop technical specifications that would solve business process issues including single sign-on, account linking and consent, among others.

The SAML v1.x specifications and the Liberty Alliance Project specifications do not compete with one another. They are complementary. In fact, the Liberty Alliance Project specifications

leverage profiles from the SAML specifications. The decision of whether to use SAML v1.x or the Liberty specifications depends on your goal. In general, SAML v1.x should suffice for single sign-on basics. The Liberty Alliance Project specifications can be used for more sophisticated functions and capabilities, such as global sign-out, attribute sharing, web services. The following table compares the benefits of the two.

TABLE 12–1 Comparison of the SAML v1.x and Liberty Alliance Project Specifications

SAML v1.x Uses	Liberty Alliance Project Uses
Cross-domain single sign-on	Single sign-on *only* after user federation
No user federation	User federation
No privacy control, best for use within one company	Built on top of SAML
User identifier is sent in plain text	User identifier is sent as a unique handle

Note – OpenSSO Enterprise has appropriated the terms from the Liberty ID-FF for all federation protocol implementations in the OpenSSO Enterprise console.

The Liberty ID-FF and SAML v1.x should only be used when integrating with a partner that is not able to use SAML v2. If you are deploying OpenSSO Enterprise with Microsoft Active Directory with Federation Services, you **must** use WS-Federation. More information on these options can be found in the following sections:

- "Using SAML" on page 156
- "Using the Liberty ID-FF" on page 164
- "Using WS-Federation" on page 174

Using SAML

SAML defines an XML-based framework for exchanging identity information across security domains for purposes of authentication, authorization and single sign-on. It was designed to be used within other specifications (the Liberty Alliance Project, the Shibboleth project, and the Organization for the Advancement of Structured Information Standards have all adopted aspects of SAML) although the latest release (SAML v2) has incorporated back into the framework elements from the specifications developed by those very same organizations. The SAML specifications consist of a number of components, illustrated by Figure 12–1.

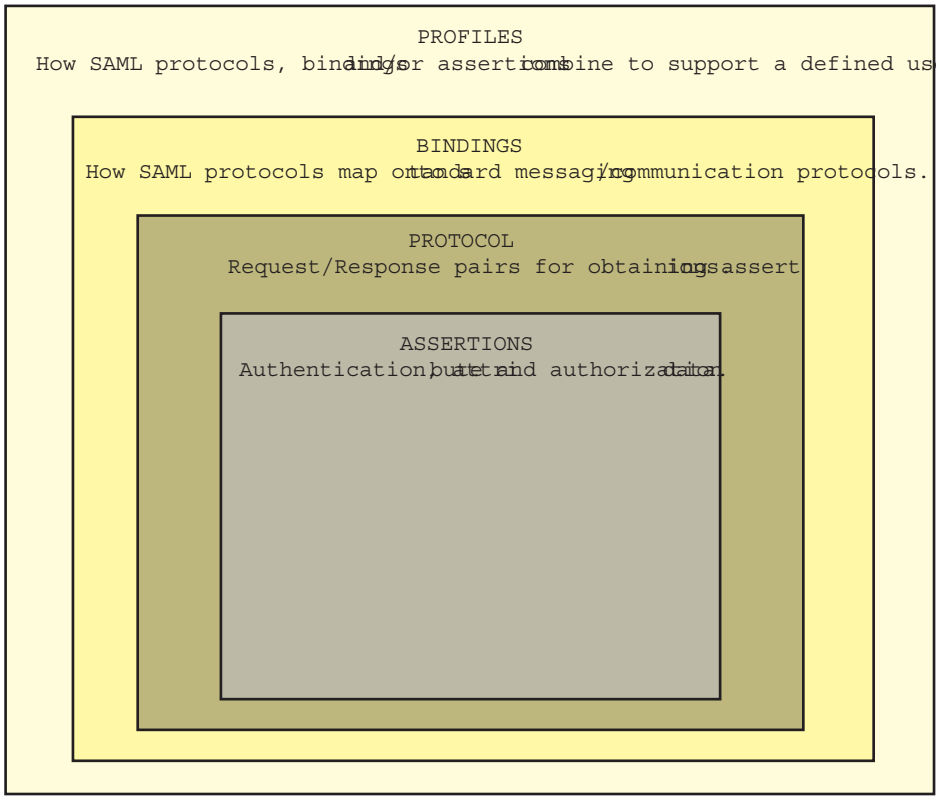


FIGURE 12–1 Components of the SAML Specifications

The SAML specification defines the *assertion* security token format as well as *profiles* that standardize the HTTP exchanges required to transfer XML requests and responses between an asserting authority and a trusted partner. An *assertion* is a package of verified security information that supplies one or more statements concerning a principal's authentication status, access authorization decisions, or identity attributes. (A person identified by an email address is a principal as might be a printer.) Assertions are issued by an asserting authority (a platform or application that declares whether a subject has been authenticated into its system), and received by relying parties (partner sites defined by the authority as *trusted*). Asserting authorities use different sources to configure assertion information, including external data stores or assertions that have already been received and verified.

The most recent SAML v2 specifications are defined more broadly than those developed for SAML v1.x — with particular attention paid to functionality dealing with federation. Before SAML v2 was introduced, SAML v1.x was simply a way to exchange identity data. In fact, up to version 1.1, the Liberty Alliance Project Identity Federation Framework (Liberty ID-FF) was developed using the SAML 1.0 specification. Liberty ID-FF version 1.2 was also developed using the SAML v1.1 specification. But, following the release of version 1.2, the Liberty ID-FF was

incorporated into the SAML v2 specification. Additionally, SAML v2 adds components of the Shibboleth initiative. Thus, SAML v2 is a major revision, providing significant additional functionality and making the previous versions of SAML incompatible with it. Going forward, SAML v2 will be the basis on which OpenSSO Enterprise implements federation. Figure 12–2 illustrates the convergence.

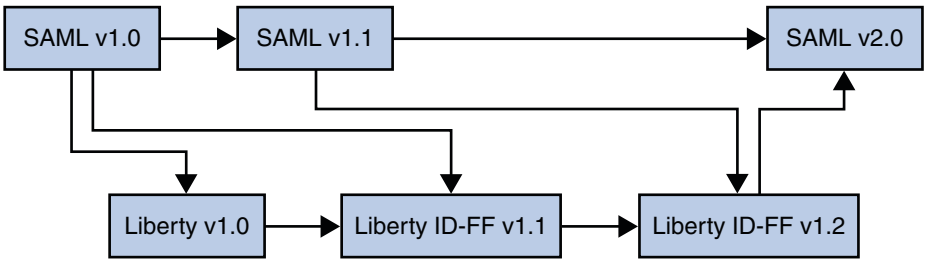


FIGURE 12–2 Liberty ID-FF and SAML Convergence

Note – For more information on this convergence (including how the Shibboleth Project was also integrated), see the Federation section of Strategic Initiatives on the Liberty Alliance Project web site.

More information on the SAML implementations can be found in the following sections).

- "About SAML v2" on page 158
- "About SAML v1.x" on page 162
- "Using SAML or OpenSSO Enterprise CDSSO" on page 164

Caution – SAML v1.x and SAML v2 assertions and protocol messages are incompatible.

About SAML v2

OpenSSO Enterprise delivers a solution that allows businesses to establish a framework for sharing trusted information across a distributed network of partners using the standards-based SAML v2. Towards this end, HTTP(S)-based service endpoints and SOAP service endpoints are supplied as well as assertion and protocol object manipulating classes. A web browser can access all HTTP(S)-based service endpoints and an application can make use of the SOAP endpoints and API as long as metadata for each participating business on BOTH sides of the SAML v2 interaction is exchanged beforehand.

Figure 12–3 illustrates the SAML v2 framework which consists of web-based services [using SOAP, XML over HTTP(S) or HTML over HTTP(S)], and Java™-based application provider interfaces (API) and service provider interfaces (SPI). Additionally, the figure shows an agent embedded into a web container in which a service provider application is deployed. This agent enables the service provider to participate in the SAML v1.x or Liberty ID-FF protocols.

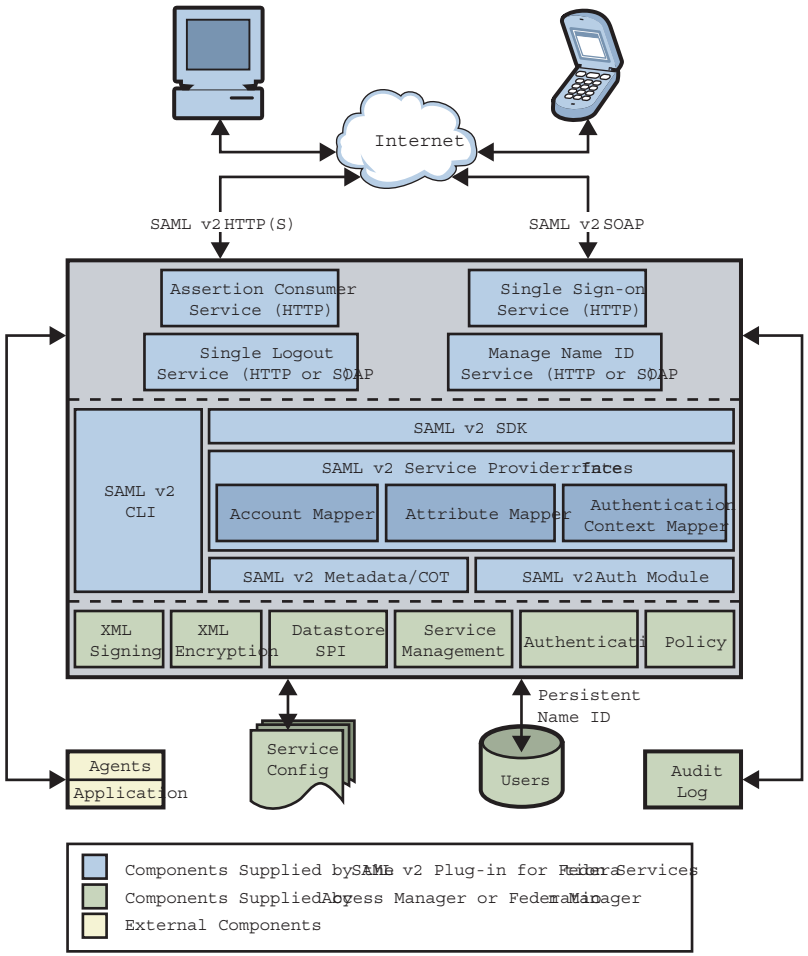


FIGURE 12–3 SAML v2 Architecture

The following sections contain more information about the SAML v2 framework.

- "Key Features" on page 160
- "Administration" on page 161
- "Application Programming Interfaces" on page 161

- "Service Provider Interfaces" on page 161
- "JavaServer Pages" on page 162

Key Features

The key features of SAML v2 in OpenSSO Enterprise include:

- Single sign-on using the POST profile, the Artifact binding (also referred to as HTTP redirect), and unsolicited responses (initiated by the identity provider)
- Single logout using HTTP redirect and SOAP binding
- Federation termination using HTTP redirect and SOAP binding
- Auto-federation (automatic linking of service provider and identity provider user accounts based on a common attribute)
- Bulk federation
- Dynamic creation of user accounts
- One time federation (transient NameID format for SSO)
- Basic Authentication, SSL and SSL with client authentication for SOAP binding security
- SAML v2 authentication
- Identity provider discovery
- XML verification, signing, encryption and decryption
- Profile initiation and processing using included JavaServer Pages™ (JSP™)
- Load balancing support
- Assertion failover
- Enhanced Client or Proxy (ECP) support in SP and IDP
- Assertion queries and requests
- Attribute queries
- New Name Identifier
- Affiliation
- Name Identifier Mapping
- XACML profile for authorization

Note – See "XACML Service" on page 133 for more information.

- Protocol coexistence with the SAML v1.x and the Liberty ID-FF

Additionally, OpenSSO Enterprise has received high scores and passed the Liberty Alliance Project interoperability tests for SAML v2. For more information, see the SAMLv2 support matrix on the Liberty Alliance Project web site.

Administration

In order to communicate using the SAML v2 profiles you need, at least, two instances of OpenSSO Enterprise. One instance will act for the identity provider and the other will act for the service provider. Name identifiers are used to communicate regarding a user.

Note – SAML v2 single sign-on interactions support both *persistent* and *transient* identifiers. A persistent identifier is saved to a particular user entry as the value of two attributes. A transient identifier is temporary and no data will be written to the user's data store entry.

To prepare your instances for SAML v2 interactions, you need to exchange a particular provider's configuration information or *metadata* between all participating identity and service providers, and assemble the providers into a *circle of trust*. Utility APIs can then be used to communicate with the data store, reading, writing, and managing the relevant properties and property values. For more information see the Chapter 10, "SAMLv2 Administration," in *Sun Federated Access Manager 8.0 Administration Guide*.

Application Programming Interfaces

The SAML v2 framework contains API that can be used to construct and process assertions, requests, and responses. The SAML v2 Java API packages include:

- The `com.sun.identity.saml2.assertion` package provides interfaces to construct and process SAML v2 assertions. It also contains the `AssertionFactory`, a factory class used to obtain instances of the objects defined in the assertion schema.
- The `com.sun.identity.saml2.common` package provides interfaces and classes used to define common SAML v2 utilities and constants.
- The `com.sun.identity.saml2.protocol` package provides interfaces used to construct and process the SAML v2 requests and responses. It also contains the `ProtocolFactory`, a factory class used to obtain object instances for concrete elements in the protocol schema.

More information can be found in "Using the SAML v2 SDK" in *Sun Federated Access Manager 8.0 Developer's Guide* and the *Federated Access Manager 8.0 Java API Reference*.

Service Provider Interfaces

The `com.sun.identity.saml2.plugins` package provides pluggable interfaces to implement SAML v2 functionality into your application. Default implementations are provided, but a customized implementation can be plugged in by modifying the corresponding attribute in the provider's extended metadata configuration file. The interfaces include mappers for:

- Account mapping (map between the account referred to in the incoming request and the local user account)

- Attribute mapping (specifies which set of user attributes in an identity provider user account needs to be included in an assertion, and maps the included attributes to attributes in the user account defined by the service provider)
- Authentication context mapping (map between Authentication Contexts defined in the SAML v2 specifications and authentication framework schemes defined in OpenSSO Enterprise (user/module/service/role/level based authentication)

More information can be found in "Service Provider Interfaces" in *Sun Federated Access Manager 8.0 Developer's Guide* and the *Federated Access Manager 8.0 Java API Reference*.

JavaServer Pages

The SAML v2 framework provides JSP that can be used to initiate single sign-on, single logout and termination requests from either the identity provider or the service provider using a web browser. The JSP accept query parameters to allow flexibility in constructing SAML v2 requests; they can be modified for your deployment. More information can be found in "JavaServer Pages" in *Sun Federated Access Manager 8.0 Developer's Guide*.

About SAML v1.x

OpenSSO Enterprise can be configured to use SAML v1.x to achieve interoperability between vendor platforms that provide SAML v1.x assertions. Assertions are issued by a SAML v1.x asserting authority (a platform or application that declares whether a subject has been authenticated into its system), and received by relying parties (partner sites defined by the authority as *trusted*). SAML v1.x authorities use different sources to configure the assertion information, including external data stores or assertions that have already been received and verified. SAML v1.x can be used to allow OpenSSO Enterprise to:

- Authenticate users and access trusted partner sites without having to reauthenticate; in effect, single sign-on.
- Act as a policy decision point (PDP), allowing external applications to access user authorization information for the purpose of granting or denying access to their resources. For example, employees of an organization can be allowed to order office supplies from suppliers if they are authorized to do so.
- Act as both an attribute authority that allows trusted partner sites to query a subject's attributes, and an authentication authority that allows trusted partner sites to query a subject's authentication information.
- Validate parties in different security domains for the purpose of performing business transactions.
- Build Authentication, Authorization Decision, and Attribute Assertions using the SAML v1.x API.
- Permit an XML-based digital signature signing and verifying functionality to be plugged in.

Note – Although Liberty ID-FF (as described in "Using the Liberty ID-FF" on page 164) integrates aspects of the SAML v1.x specifications, its usage of SAML v1.x is independent of the SAML v1.x framework as described in this section.

Figure 12–4 illustrates how SAML v1.x interacts with the other components in OpenSSO Enterprise.

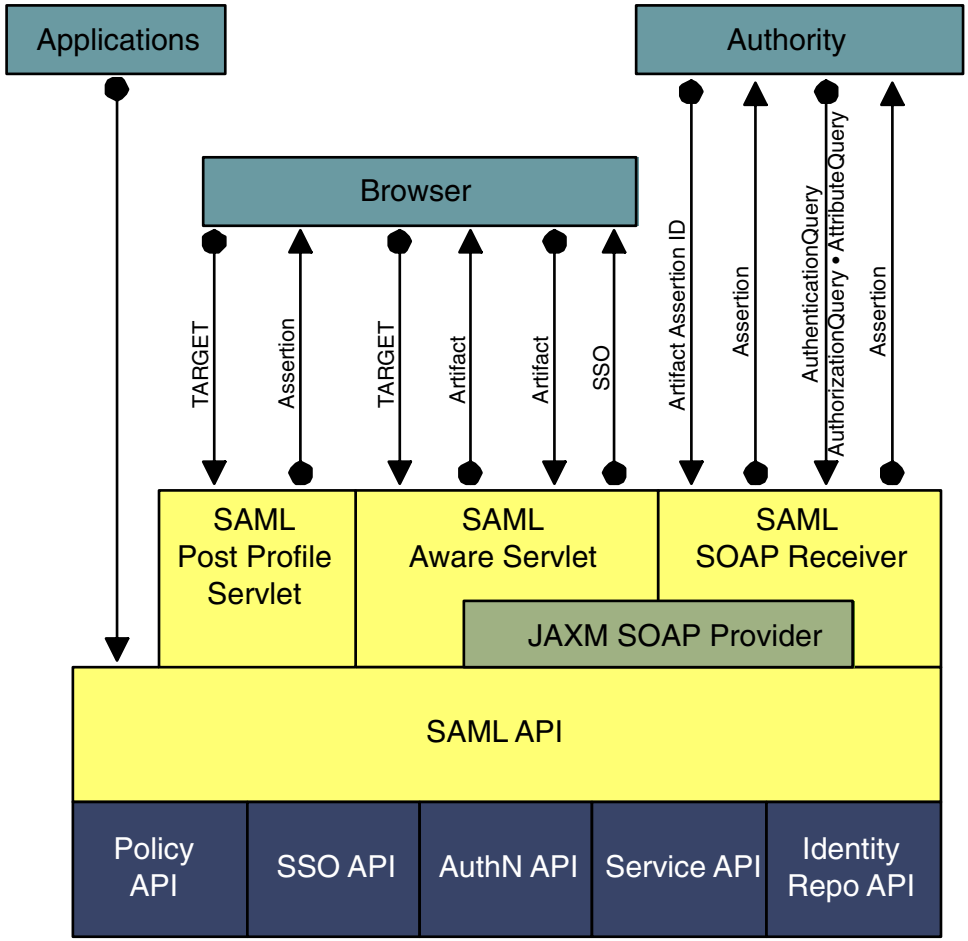


FIGURE 12–4 SAML v1.x Interaction in OpenSSO Enterprise

Using SAML or OpenSSO Enterprise CDSSO

Cross Domain Single Sign On (CDSSO) is a proprietary mechanism from Sun OpenSSO Enterprise, designed before SAML and the Liberty Alliance Project specifications existed. CDSSO is still available and, in certain cases, is easier to set up and manage than the latter two specifications — which solve a broader set of SSO issues than CDSSO. CDSSO requires all policy agents to be configured to use a single OpenSSO Enterprise server. This means only one user identity can exist in the entire system whereas, when using SAML/Liberty Alliance Project, user identities can exist on multiple systems (service providers or identity providers). Because of the single identity in CDSSO interactions, issues such as account mapping, attribute flow and session synchronization are not relevant thus, if you need to implement these features, use SAML. If the following points are valid to your planned deployment, CDSSO may be a simpler and more suitable solution than SAML/Liberty Alliance Project:

- Only Sun OpenSSO Enterprise and Sun policy agents are involved.
- Sun policy agents are configured to use the same OpenSSO Enterprise infrastructure where multiple instances can exist.
- OpenSSO Enterprise uses a single user identity store.
- Multiple instances of OpenSSO Enterprise (configured for high-availability) must reside in a single DNS domain. Only policy agents can reside in different DNS domains.

For more information on CDSSO, see Chapter 7, "Models of the User Session and Single Sign-On Processes."

Using the Liberty ID-FF

The Liberty Alliance Project was formed to develop technical specifications that would solve business process issues including single sign-on, federation and consent. The Liberty Alliance Project Identity Federation Framework (Liberty ID-FF) uses a name identifier to pass identity data between identity providers and service providers. The *name identifier* is a randomly generated character string that is assigned to a principal and used to federate the principal's accounts at the identity provider and service provider sites. This pseudonym allows all providers to identify a principal without knowing the user's actual identity. The name identifier has meaning only in the context of the relationship between providers. SAML v1.x is used for provider interaction.

Note – Liberty ID-FF was initially defined as an extension of SAML 1.0 (and later SAML 1.1). The extensions have now been contributed back into SAML v2 which, going forward, will be the basis on which the Liberty Alliance Project builds additional federated identity applications. See "Using SAML" on page 156 for more information on this convergence.

The following sections contain information about the Liberty ID-FF and the features implemented in OpenSSO Enterprise.

Liberty ID-FF Features

The following sections contain information about the Liberty ID-FF features implemented in OpenSSO Enterprise.

Federated Single Sign-On

Let's assume that a principal has separate user accounts with a service provider and an identity provider in the same circle of trust. In order to gain access to these individual accounts, the principal would authenticate with each provider separately. If federating with the Liberty ID-FF though, after authenticating with the service provider, the principal may be given the option to federate the service provider account with the identity provider account. Consenting to the federation of these accounts links them for SSO, the means of passing a user's credentials between applications without the user having to reauthenticate. SSO and federated SSO have different processes. With OpenSSO Enterprise, you can achieve SSO in the following ways:

- Install a policy agent in a web container to protect the application and pass the `HTTP_HEADER` and `REMOTE_USER` variables to the application to capture the user credentials. You may or may not need a custom authentication module.
- Customize the application's authentication module to create an `SSOToken` from the request object or from the SSO cookie. Afterwards, retrieve the user credentials using the SSO API and create a session data structure using the application's API.

To set up federated SSO, you must first establish SSO. Following that, enable federation in the metadata for the service provider entity and the identity provider entity using OpenSSO Enterprise. Liberty ID-FF providers differentiate between federated users by defining a unique *identifier* for each account. (They are not required to use the principal's actual provider account identifier.) Providers can also choose to create multiple identifiers for a particular principal. However, identity providers must create one handle per user for service providers that have multiple web sites so that the handle can be resolved across all of them.

Note – Because both the identity provider entity and the service provider entity in a federation need to remember the principal's identifier, they create entries that note the value in their respective user repositories. In most scenarios, the identity provider's identifier is conveyed to a service provider and not visa versa. For example, if a service provider does not maintain its own user repository, the identity provider's identifier is used.

OpenSSO Enterprise can accommodate the following SSO and federation-related functions:

- Providers of either type notify the principal upon identity federation or defederation.
- Providers of either type notify each other regarding a principal's defederation.
- Identity providers notify the appropriate service providers regarding a principal's account termination.
- Providers of either type display a list of federated identities to the principal.
- Users can terminate federations or defederate identities.

Additionally, OpenSSO Enterprise can accommodate the federation features explained in the following sections.

- "Auto-Federation" on page 166
- "Bulk Federation" on page 166

Auto-Federation

Auto federation will automatically federate a user's disparate provider accounts based on a common attribute. During SSO, if it is deemed a user at provider A and a user at provider B have the same value for the defined common attribute (for example, an email address), the two accounts will be federated without consent or interaction from the principal. For more information, see "Auto-Federation" in *Sun Federated Access Manager 8.0 Administration Guide*.

Bulk Federation

Federating one user's service provider account with their identity provider account generally requires the principal to visit both providers and link them. An organization though needs the ability to federate user accounts behind the scenes. OpenSSO Enterprise provides a script for federating user accounts in bulk. The script allows the administrator to federate many (or all) of a principal's provider accounts based on metadata passed to the script. Bulk federation is useful when adding a new service provider to an enterprise so you can federate a group of existing employees to the new service. For more information, see "Bulk Federation" in *Sun Federated Access Manager 8.0 Administration Guide*.

Authentication and Authentication Context

SSO is the means by which a provider of either type can convey to another provider that a principal has been authenticated. Authentication is the process of validating user credentials; for example, a user identifier accompanied by an associated password. You can authenticate users with OpenSSO Enterprise in the following ways:

- Use a policy agent to insert HTTP header variables into the request object. This functions for web applications only.
- Use the authentication API to validate and retrieve user identity data. This will work with either web or non-web applications.

Identity providers use local (to the identity provider) session information mapped to a user agent as the basis for issuing SAML authentication assertions to service providers. Thus, when the principal uses a user agent to interact with a service provider, the service provider requests authentication information from the identity provider based on the user agent's session information. If this information indicates that the user agent's session is presently active, the identity provider will return a positive authentication response to the service provider. OpenSSO Enterprise allows providers to exchange the following minimum set of authentication information with regard to a principal.

- Authentication status (active or not)
- Instant (time authenticated)
- Authentication method
- Pseudonym (temporary or persistent)

SAML v1.x is used for provider interaction during authentication but not all SAML assertions are equal. Different authorities issue SAML assertions of different quality. Therefore, the Liberty ID-FF defines how the consumer of a SAML assertion can determine the amount of assurance to place in the assertion. This is referred to as the *authentication context*, information added to the SAML assertion that gives the assertion consumer the details they need to make an informed entitlement decision. For example, a principal uses a simple identifier and a self-chosen password to authenticate to a service provider. The identity provider sends an assertion to a second service provider that states how the principal was authenticated to the first service provider. By including the authentication context, the second service provider can place an appropriate level of assurance on the associated assertion. If the service provider were a bank, they might require stronger authentication than that which has been used and respond to the identity provider with a request to authenticate the user again using a more stringent context. The authentication context information sent in the assertion might include:

- The initial user identification mechanism (for example, face-to-face, online, or shared secret).
- The mechanisms for minimizing compromise of credentials (for example, private key in hardware, credential renewal frequency, or client-side key generation).
- The mechanisms for storing and protecting credentials (for example, smart card, or password rules).

- The authentication mechanisms (for example, password or smart card with PIN).

The Liberty ID-FF specifications define authentication context *classes* against which an identity provider can claim conformance. The Liberty ID-FF authentication contexts are listed and described in the following table.

TABLE 12–2 Authentication Context Classes

Class	Description
MobileContract	Identified when a mobile principal has an identity for which the identity provider has vouched.
MobileDigitalID	Identified by detailed and verified registration procedures, a user's consent to sign and authorize transactions, and DigitalID-based authentication.
MobileUnregistered	Identified when the real identity of a mobile principal has not been strongly verified.
Password	Identified when a principal authenticates to an identity provider by using a password over an unprotected HTTP session.
Password-ProtectedTransport	Identified when a principal authenticates to an identity provider by using a password over an SSL-protected session.
Previous-Session	Identified when an identity provider must authenticate a principal for a current authentication event and the principal has previously authenticated to the identity provider. This affirms to the service provider a time lapse from the principal's current resource access request. **Note –** The context for the previously authenticated session is not included in this class because the user has not authenticated during this session. Thus, the mechanism that the user employed to authenticate in a previous session should not be used as part of a decision on whether to now allow access to a resource.
Smartcard	Identified when a principal uses a smart card to authenticate to an identity provider.
Smartcard-PKI	Identified when a principal uses a smart card with an enclosed private key and a PIN to authenticate to an identity provider.
Software-PKI	Identified when a principal uses an X.509 certificate stored in software to authenticate to the identity provider over an SSL-protected session.

TABLE 12–2 Authentication Context Classes *(Continued)*

Class	Description
Time-Sync-Token	Identified when a principal authenticates through a time synchronization token.

For more information, see the *Liberty ID-FF Authentication Context Specification* and . Additionally, there is an XML schema defined which the identity provider authority can use to incorporate the context of the authentication in the SAML assertions it issues.

The Common Domain for Identity Provider Discovery

Service providers need a way to determine which identity provider in a circle of trust is used by a principal requesting authentication. Because circles of trust are configured without regard to their location, this function must work across DNS-defined domains. A common domain is configured, and a common domain cookie written, for this purpose.

Let's suppose a circle of trust contains more than one identity provider. In this case, a service provider trusts more than one identity provider so, when a principal needs authentication, the service provider with which the principal is communicating must have the means to determine the correct identity provider. To ascertain a principal's identity provider, the service provider invokes a protocol exchange to retrieve the *common domain cookie*, a cookie written for the purpose of *introducing* the identity provider to the service provider. If no common domain cookie is found, the service provider will present a list of trusted identity providers from which the principal can choose. After successful authentication, the identity provider writes (using the configured Writer Service URL) a common domain cookie and, the next time the principal attempts to access a service, the service provider finds and reads the common domain cookie (using the configured Reader Service URL), to determine the identity provider. More information on the Common Domain for Identity Provider Discovery is available in the following sections, and in "Common Domain Services for Federation Management" in *Sun Federated Access Manager 8.0 Administration Guide*.

- "The Common Domain" on page 169
- "The Common Domain Cookie" on page 170
- "The Writer Service and the Reader Service" on page 170

The Common Domain

The *common domain* is established for use only within the scope of identity provider discovery in a defined circle of trust. In OpenSSO Enterprise deployments, the identity provider discovery WAR is deployed in a web container installed in a predetermined and preconfigured *common* domain so that the common domain cookie is accessible to all providers in the circle of trust. For example, if an identity provider is available at `http://www.Bank.com`, a service provider is available at `http://www.Store.com`, and the defined common domain is `RetailGroup.com`, the addresses will be `Bank.RetailGroup.com` and `Store.RetailGroup.com`, respectively. If the

HTTP server in the common domain is operated by the service provider, the service provider will redirect the user agent to the appropriate identity provider.

The Common Domain Cookie

After an identity provider authenticates a principal, the identity provider sets a URL-encoded cookie defined in a predetermined domain common to all identity providers and service providers in the circle of trust. The *common domain cookie* is named `_liberty_idp` for Liberty ID-FF and `_saml_idp` for SAML v2. After successful authentication, a principal's identity provider appends their particular encoded identifier to a list in the cookie. If their identifier is already present in the list, the identity provider may remove the initial appearance and append it again. The intent is that the service provider reads the last identifier on the cookie's list to find the principal's most recently established identity provider.

Note – The identifiers in the common domain cookie are a list of `SuccinctID` elements encoded in the Base64 format. One element maps to each identity provider in the circle of trust. Service providers then use this `SuccinctID` element to find the user's preferred identity provider.

The Writer Service and the Reader Service

After a principal authenticates with a particular identity provider, the identity provider redirects the principal's browser to the configured Writer Service URL using a parameter that indicates they are the identity provider for this principal. The Writer Service then writes a cookie using the parameter. Thereafter, all providers configured in this common domain will be able to tell which identity provider is used by this principal. Thus, the next time the principal attempts to access a service hosted by a service provider in the same common domain, the service provider retrieves and reads the common domain cookie, using the configured Reader Service URL, to determine the identity provider.

The Writer Service URL and the Reader Service URL can be defined for use with the Liberty ID-FF or the SAML v2 federation protocol. The URLs are defined when you create a circle of trust for federation. The Common Domain for Identity Provider Discovery for Liberty ID-FF is based on the Identity Provider Introduction Profile detailed in the *Liberty ID-FF Bindings and Profiles Specifications*. The Common Domain for Identity Provider Discovery for SAML v2 is an implementation of the Identity Provider Discovery Profile as described in the Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0 specification.

Identifiers and Name Registration

OpenSSO Enterprise supports name identifiers that are unique across all providers in a circle of trust. This identifier can be used to obtain information for or about the principal without requiring the user to consent to a long-term relationship with the service provider. When beginning federation, the identity provider generates an opaque value that serves as the initial name identifier that both the service provider and the identity provider use to refer to the

principal when communicating with each other. After federation, the identity provider or the service provider may register a different opaque value. If a service provider registers a different opaque value for the principal, the identity provider must use the new identifier when communicating with the service provider about the principal. The reasons for changing an identifier would be implementation-specific. The initial name identifier defined by the identity provider is always used to refer to the principal unless a new name identifier is registered.

Global Logout

A principal may establish authenticated sessions with both an identity provider and individual service providers, based on authentication assertions supplied by the identity provider. When the principal logs out of a service provider session, the service provider sends a logout message to the identity provider that provided the authentication for that session. When this happen, or the principal manually logs out of a session at an identity provider, the identity provider sends a logout message to each service provider to which it provided authentication assertions under the relevant session. The one exception is the service provider that sent the logout request to the identity provider.

Dynamic Identity Provider Proxying

An identity provider can choose to proxy an authentication request to an identity provider in another authentication domain if it knows that the principal has been authenticated with this identity provider. The proxy behavior is defined by the local policy of the proxying identity provider. However, a service provider can override this behavior and choose not to proxy. This function can be implemented as a form of authentication when, for instance, a roaming mobile user accesses a service provider that is not part of the mobile home network. For more information see "Dynamic Identity Provider Proxying" in *Sun Federated Access Manager 8.0 Administration Guide*.

About the Liberty ID-FF Process

The Liberty ID-FF is designed to work with heterogeneous platforms, various networking devices (including personal computers, mobile phones, and personal digital assistants), and emerging technologies. The process of Liberty ID-FF federation begins with authentication. A user attempting to access a resource protected by OpenSSO Enterprise are redirected to the proprietary Authentication Service via an OpenSSO Enterprise login page. After the user provides credentials, the Authentication Service allows or denies access to the resource based on the outcome.

Note – For more information about the proprietary Authentication Service, see the Chapter 8, "Authentication and the Authentication Service."

When the user attempts access to a resource that belongs to a trusted member provider of a configured circle of trust, the process of user authentication begins with the search for a valid OpenSSO Enterprise session token from the proprietary Authentication Service. The process can go in one of two directions based on whether a session token is found.

- If no session token is found, the principal is redirected to a location defined by the pre-login URL to establish a valid session.
- If a session token is found, the principal is granted (or denied) access to the requested page. Assuming access is granted, the requested page contains a link so the principal may federate the OpenSSO Enterprise identity with the identity local to the requested site. If the principal clicks this link, federation begins.

Figure 12–5 illustrates these divergent paths. The process shown is the default process when no application has been deployed. When an application is deployed and using OpenSSO Enterprise, the process will change based on the query parameters and preferences passed to OpenSSO Enterprise from the participating application. For more information, see "The Pre-login URL" in *Sun Federated Access Manager 8.0 Administration Guide*.

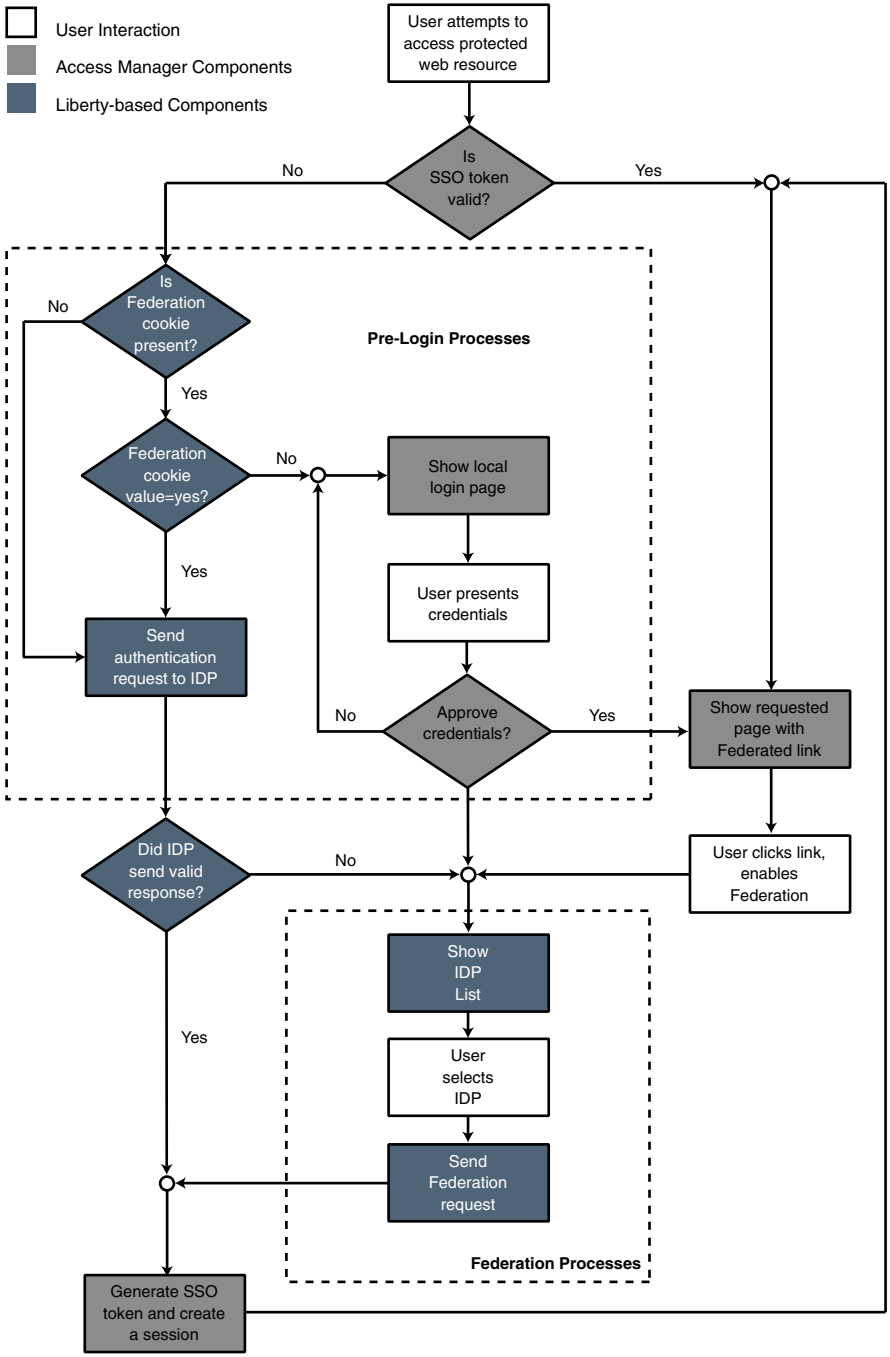


FIGURE 12–5 Default Process of Federation

As illustrated, the pre-login process establishes a valid OpenSSO Enterprise session. When a principal attempts to access a service provider site and no OpenSSO Enterprise session token is found, OpenSSO Enterprise searches for a federation cookie. A *federation cookie* is implemented by OpenSSO Enterprise and is called `fedCookie`. It can have a value of either `yes` or `no`, based on the principal's federation status.

Note – A federation cookie is *not* defined in the Liberty Alliance Project specifications.

At this point, the pre-login process may take one of the following paths:

- If a federation cookie is found and its value is `no`, a OpenSSO Enterprise login page is displayed and the principal submits credentials to the proprietary Authentication Service. When authenticated by OpenSSO Enterprise, the principal is redirected to the requested page, which might contain a link to allow for identity federation. If the principal clicks this link, federation begins.
- If a federation cookie is found and its value is `yes`, the principal has already federated an identity but has not been authenticated by an identity provider within the circle of trust for this OpenSSO Enterprise session. Authentication to OpenSSO Enterprise is achieved on the back end by sending a request to the principal's identity provider. After authentication, the principal is directed back to the requested page.
- If no federation cookie is found, a *passive* authentication request (one that does not allow identity provider interaction with the principal) is sent to the principal's identity provider. If an affirmative authentication is received back from the identity provider, the principal is directed to the OpenSSO Enterprise Authentication Service, where a session token is granted. The principal is then redirected to the requested page. If the response from the identity provider is negative (for example, if the session has timed out), the principal is sent to a common login page to complete either a local login or Liberty ID-FF federation.

Using WS-Federation

WS-Federation is part of the larger Web Services Security (WS-Security) framework which provides a means for applying security to web services through the use of security tokens. WS-Security describes how to attach signature and encryption headers as well as security tokens (including binary security tokens such as X.509 certificates and Kerberos tickets) to SOAP messages. WS-Trust, another specification in the WS-Security framework, provides for federation by defining a Security Token Service (STS) and a protocol for requesting and issuing the security tokens. WS-Federation, as implemented in OpenSSO Enterprise, uses the OpenSSO Enterprise Security Token Service (modelled on the WS-Trust specification) to allow providers in different security realms to broker trust using information on identities, identity attributes and authentication, and provider federation. A principal requests a token from the Security Token Services. This token, which may represent the principal's primary identity, a pseudonym, or the appropriate attributes, is presented to the service provider for

authentication and authorization. WS-Federation uses several security tokens as well as the mechanism for associating them with messages. This release of OpenSSO Enterprise has implemented the following features of the WS-Federation specification.

- The *Web (Passive) Profile* defines single sign-on, single logout, attribute and pseudonym token exchanges for passive requestors; for example, a web browser that supports HTTP. For the passive mechanisms to provide a single or reduced sign-on, there needs to be a service that will verify that the claimed requestor is really the requestor. Initial verification MUST occur in a secure environment; for example, using SSL/TLS or HTTP/S. The token is abstract and the token exchange is based on the Security Token Service model of WS-Trust.
- Tokens based on the *Web Services-Interoperability Basic Security Profile* (WS-I BSP) define security that is implemented inside a SOAP message; and security implemented at the transport layer, using HTTPS. The protocol covers how you generate or handle security tokens.

The WS-Federation implementation in OpenSSO Enterprise is based on the application's SAML v2 code and uses WS-Federation 1.1 metadata. Authentication request parameters are represented directly as GET parameters, and the authentication response is a WS-Trust `RequestSecurityTokenResponse` element.

Note – There is no authentication context mapping, persistent or transient NameID identifiers or auto-federation in the OpenSSO Enterprise implementation of WS-Federation.

The entry points for all WS-Federation functionality will be implemented as servlets. JavaServer Pages (JSP) are used only for HTML content (for example, the HTML form used to send the WS-Federation single response from the identity provider to the service provider). The following figure illustrates the flow of messages between OpenSSO Enterprise (acting as the service provider) and the Active Directory (acting as the identity provider).

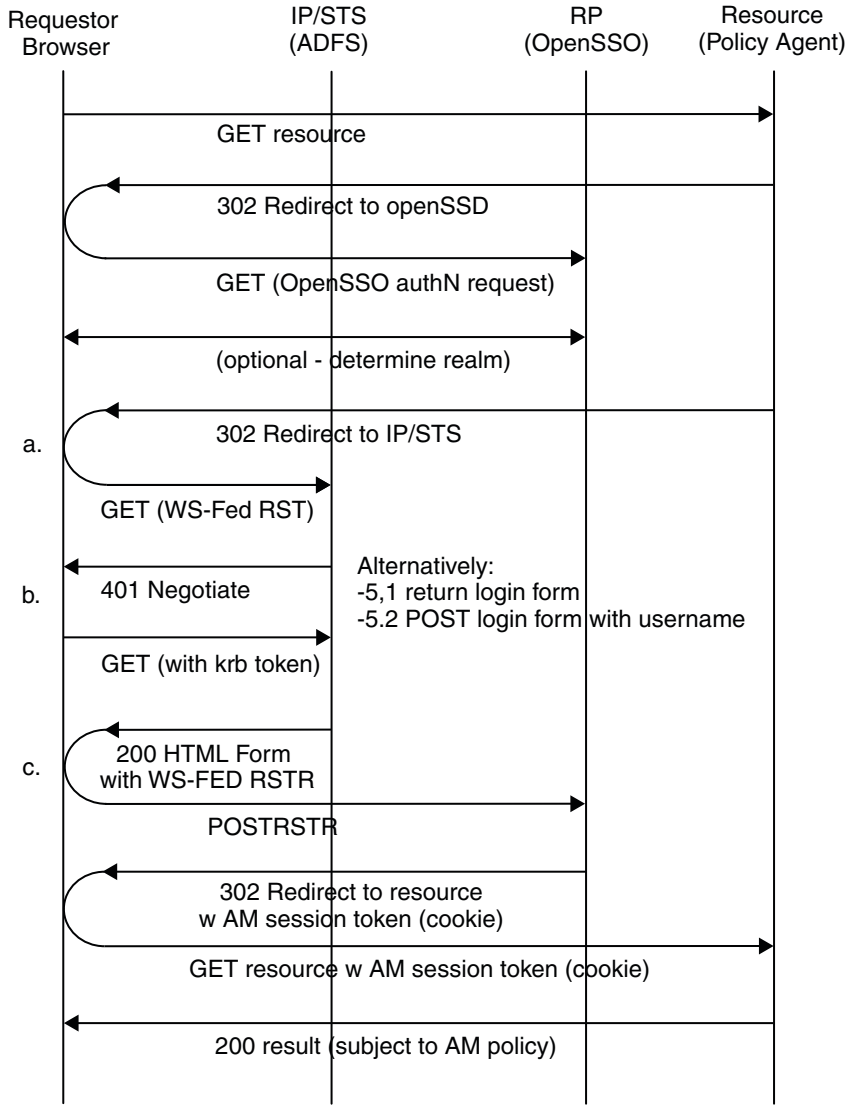


Figure 1: WS-Federation SSO Message Flow

FIGURE 12–6 WS-Federation Process Flow

In a WS-Federation interaction, the request by an unauthenticated user attempting to access a protected web site is redirected to the Active Directory for Federation Services (ADFS) identity provider. After the user is authenticated (either by a back-end single sign-on or by entering

credentials), ADFS posts a form containing a signed SAML assertion to the service provider. The service provider validates the assertion, copies the attributes into the user's session, and gives the appropriate access.

Note – Microsoft Active Directory Federation Services supports single sign-on via WS-Federation.

PART IV

The Web Services Stack, Identity Services, and Web Services Security

This fourth part of the Sun OpenSSO Enterprise Technical Overview contains information on implementing the Web Services Stack, identity services, and web services security. It contains the following chapters:

◆ ◆ ◆ CHAPTER 13

Accessing the Web Services Stack

In OpenSSO Enterprise, the Federation Services framework enables the secure exchange of authentication and authorization information by providing an interface for creating, modifying, and deleting circles of trust and configuring service providers and identity providers (both remote and hosted types) as entity providers. The implemented Web Services Stack defines a supporting framework for these Federation Services. This chapter contains information on the Web Services Stack.

About the Web Services Stack

Web services are distributed applications developed using open technologies such as eXtensible Markup Language (XML), SOAP, and HyperText Transfer Protocol (HTTP). Enterprises use these technologies as a mechanism for allowing their applications to cross network boundaries and communicate with those of their partners, customers and suppliers. Towards this end, OpenSSO Enterprise implements the Liberty Alliance Project Identity-Web Service Framework (Liberty ID-WSF) 1.1 specifications, designed to operate in concert with the Liberty Alliance Project Identity-Federation Framework (Liberty ID-FF). The implementation of the Liberty ID-WSF 1.1 specifications uses a servlet framework into which identity-based web services can be plugged and leveraged for security. Tools and API are also provided for identity providers to develop new web services and for service providers to consume both default and custom web services. Furthermore, OpenSSO Enterprise provides the necessary hooks to integrate an existing enterprise infrastructure with the Liberty Alliance Project—based infrastructure. Figure 13–1 illustrates the design of the Liberty ID-WSF framework.

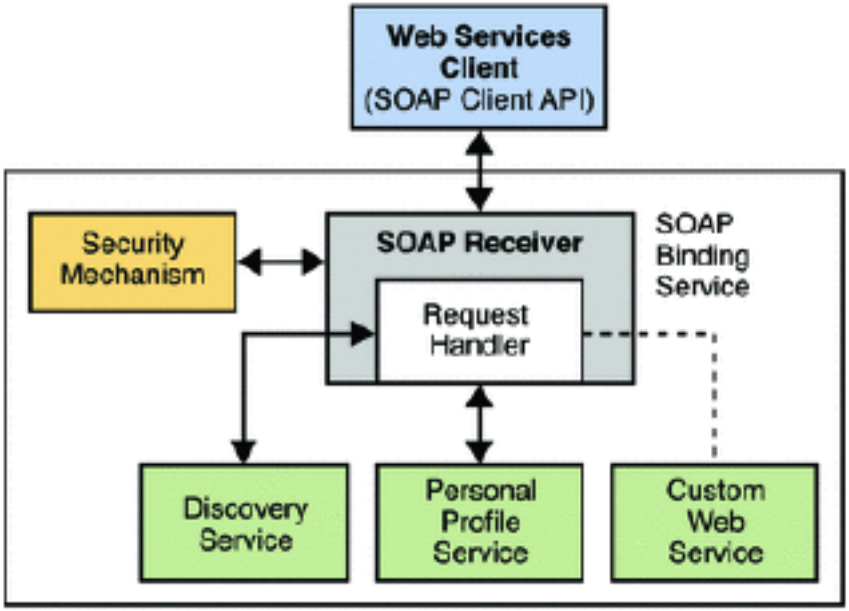


FIGURE 13–1 Web Services Stack Design

Any custom web service developed with the Liberty ID-WSF must register with the SOAP Binding Service which provides validation of SOAP messages and generates the OpenSSO Enterprise session token for client authorization. Bootstrapping of the Web Services Stack is accomplished by using the Discovery Service to find a *resource offering* for an authenticated user. (A *resource offering* defines an association between a type of identity data and a URI to a WSDL file that provides information about obtaining access to the data. The web service provider must also register the web service's resource offering with the Discovery Service before offering services.) OpenSSO Enterprise supports bootstrapping using SAML v2, Liberty ID-FF, or the Authentication Web Service.

Web Services Stack Architecture

Remark 13–1 Reviewer **Please explain this: Approach: Scenario 1: WSC uses the resource offering configured in discovery service to determine the version. Scenario 2: Stand alone WSC uses the version configured in the AMConfig.properties Scenario 3: WSP/Discovery determines the version to be used in response is based on in-coming message request version. If TLS:null is used, it uses the default version as always be WSF1.1**

The Web Services Stack defines an architecture in which SOAP over HTTP(S) is used as the transport layer protocol. As well, custom web services can be plugged into it. All web services in OpenSSO Enterprise (whether proprietary or custom) are front-ended by a servlet endpoint

called the `SOAPReceiver`. The `SOAPReceiver` validates digital signatures or encryptions from incoming SOAP request messages and authenticates the remote web services client. From a high-level, a user requests a specific service which passes the request to the `SOAPReceiver` which, in turn, passes it to the Liberty Personal Profile Service (or a custom web service). Figure 13–2 illustrates the architecture of the Web Services Stack and how a web service client (WSC) communicates with the web service provider (WSP).

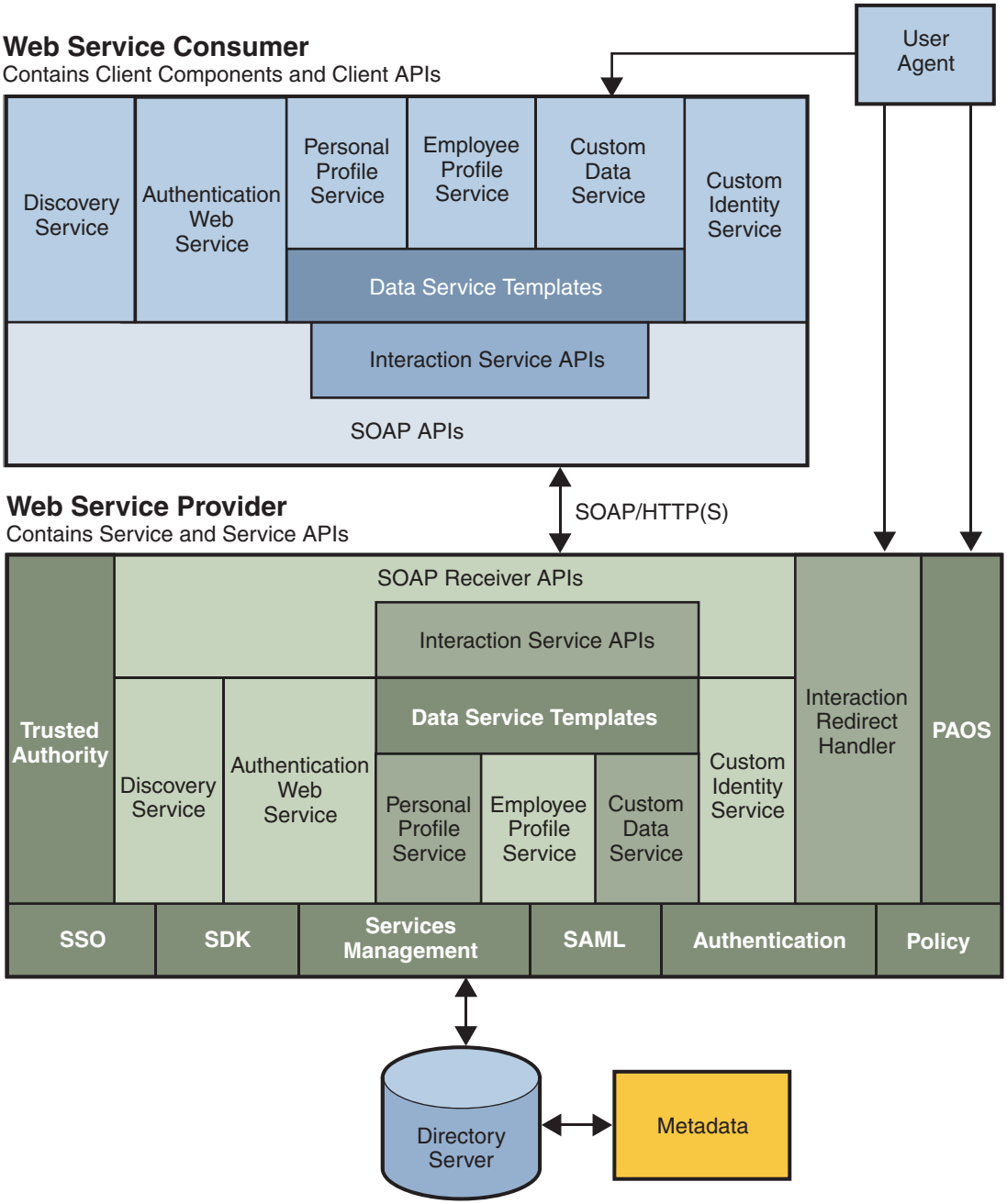


FIGURE 13–2 Web Services Stack Architecture

Web Services Stack Process

Figure 13–3 provides a high-level view of the process between the various components in the Web Services Stack. In this example:

- The web browser represents a user.
- The service provider also acts as a WSC, invoking a web service on behalf of the user. The service provider relies on the identity provider for authentication.
- The identity provider acts as an authentication provider by authenticating the user. It also acts as a trusted authority, issuing security tokens through the Discovery Service.
- The WSP serves requests from web services clients such as the Liberty Personal Profile Service.
- The process assumes that the user, the identity provider, and the service provider have already been federated.

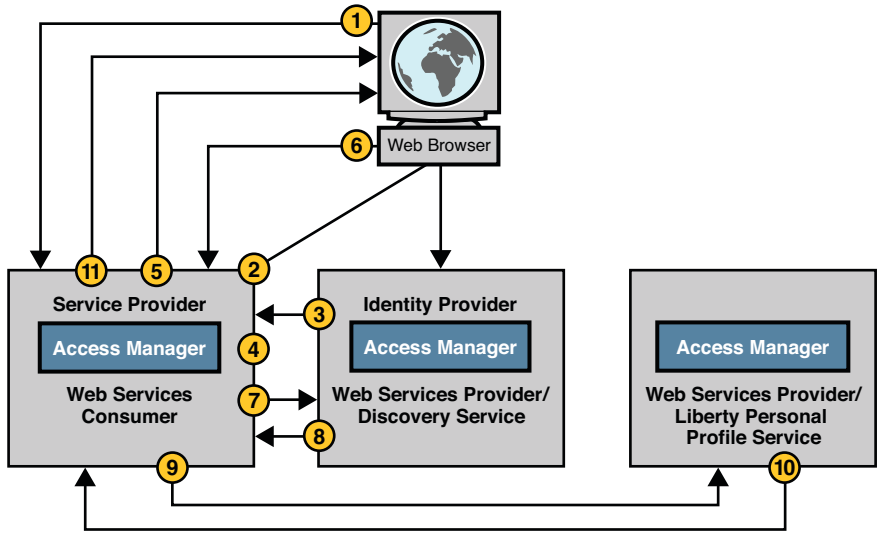


FIGURE 13–3 Web Services Stack Process

1. The user attempts to access a resource hosted on the service provider server.
2. The service provider redirects the user to the identity provider for authentication.
3. The identity provider authenticates the user successfully and sends the single sign-on assertion to the requesting service provider.
4. The service provider verifies the assertion and the user is issued a session token.
5. The service provider redirects the user to the requested resource.

6. The user requests access to another service hosted on the WSC server.

   For example, it might need that value of an attribute from the user's Liberty Personal Profile Service.

7. The WSC sends a query to the Discovery Service to determine where the user's Liberty Personal Profile Service instance is hosted.

   The WSC bootstraps the Discovery Service with the resource offering from the assertion obtained earlier.

8. The Discovery Service returns a response to the WSC containing the endpoint for the user's Liberty Personal Profile Service instance and a security token that the WSC can use to access it.

9. The WSC sends a query to the Liberty Personal Profile Service instance.

   The query asks for the user's personal profile attributes, such as home phone number. The required authentication mechanism specified in the Liberty Personal Profile Service resource offering must be followed.

10. The Liberty Personal Profile Service instance authenticates and validates authorization for the requested user or the WSC, or both.

    If user interaction is required for some attributes, the Interaction Service will be invoked to query the user for consents or for attribute values. The Liberty Personal Profile Service instance returns a response to the WSC after collecting all required data.

11. The WSC processes the Liberty Personal Profile Service response, and renders the service pages containing the information.

For detailed information about all these components, see the XXXXXX Sun Java System Access Manager 7.1 Federation and SAML Administration Guide.

Using the Web Services Stack

The Web Services Stack authenticates the user and obtains bootstrapping information for the requested application. For client applications accessed from a desktop, the user can be authenticated using the Authentication Web Service. (For example, a Java® Swing client application can authenticate the user, obtain profile data, and pass it on for online transactions.) For browser-based applications, the user can be authenticated using the SAML v2 single sign-on profiles or Liberty ID-FF. The following sections describe how the Web Services Stack can be implemented.

- "With SAML v2 or Liberty ID-FF" on page 187
- "With the Authentication Web Service" on page 189

For more information on configuring the Web Services Stack, see Chapter 8, "Implementing Web Services," in *Sun Federated Access Manager 8.0 Developer's Guide* and Chapter 8, "Web Services," in *Sun Federated Access Manager 8.0 Administration Guide*.

With SAML v2 or Liberty ID-FF

OpenSSO Enterprise can be a deployed as a service provider or an identity provider and provide identity authentication using the SAML v2 or Liberty ID-FF protocols, implemented by the Federation Services, to bootstrap into the Web Services Stack framework. The SAML v2 process is illustrated in Figure 13–4.

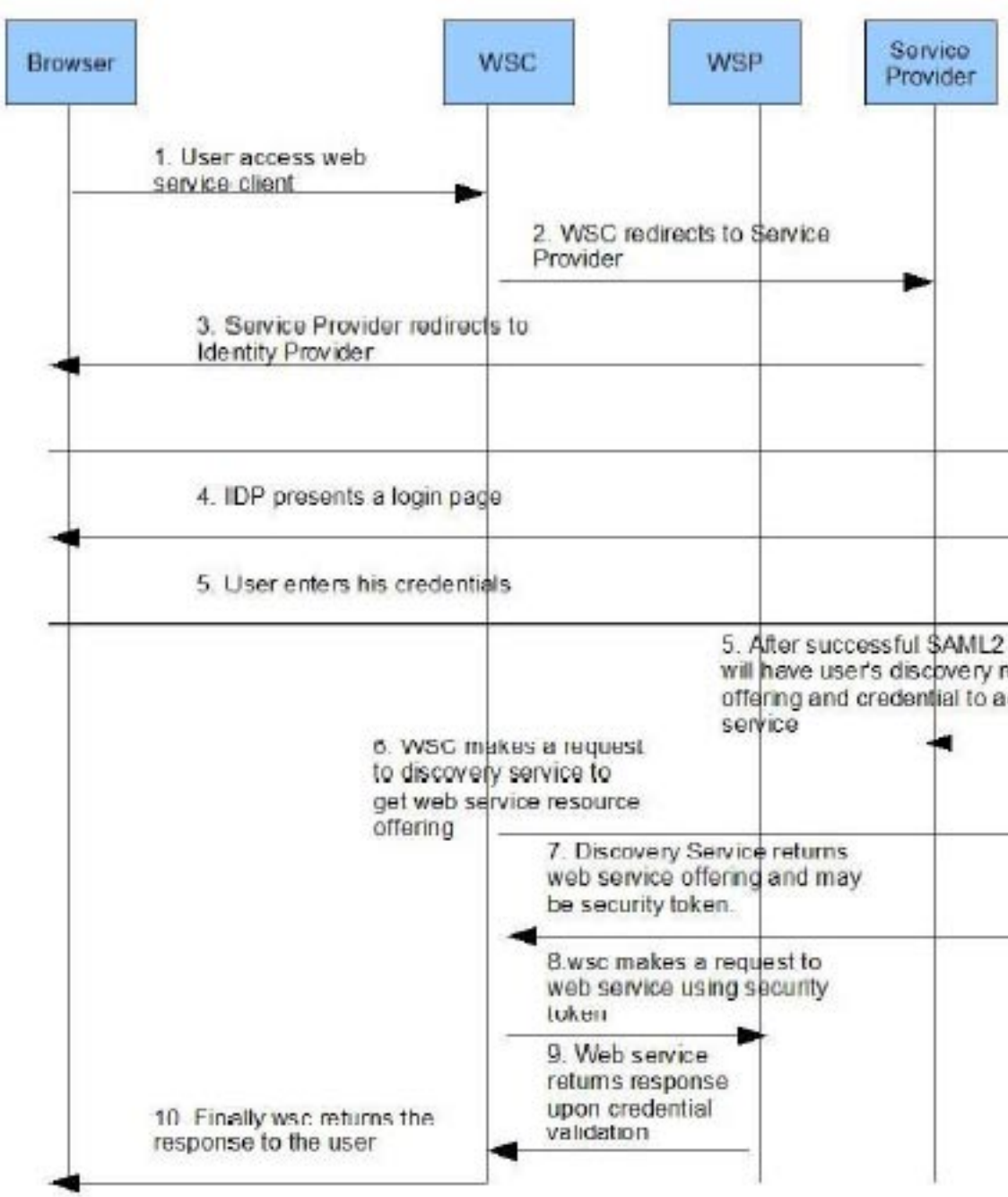


FIGURE 13–4 Web Services Stack Using SAML v2

With the Authentication Web Service

OpenSSO Enterprise can also provide identity authentication using the Authentication Web Service to bootstrap into the Web Services Stack framework. This process is illustrated in Figure 13–5.

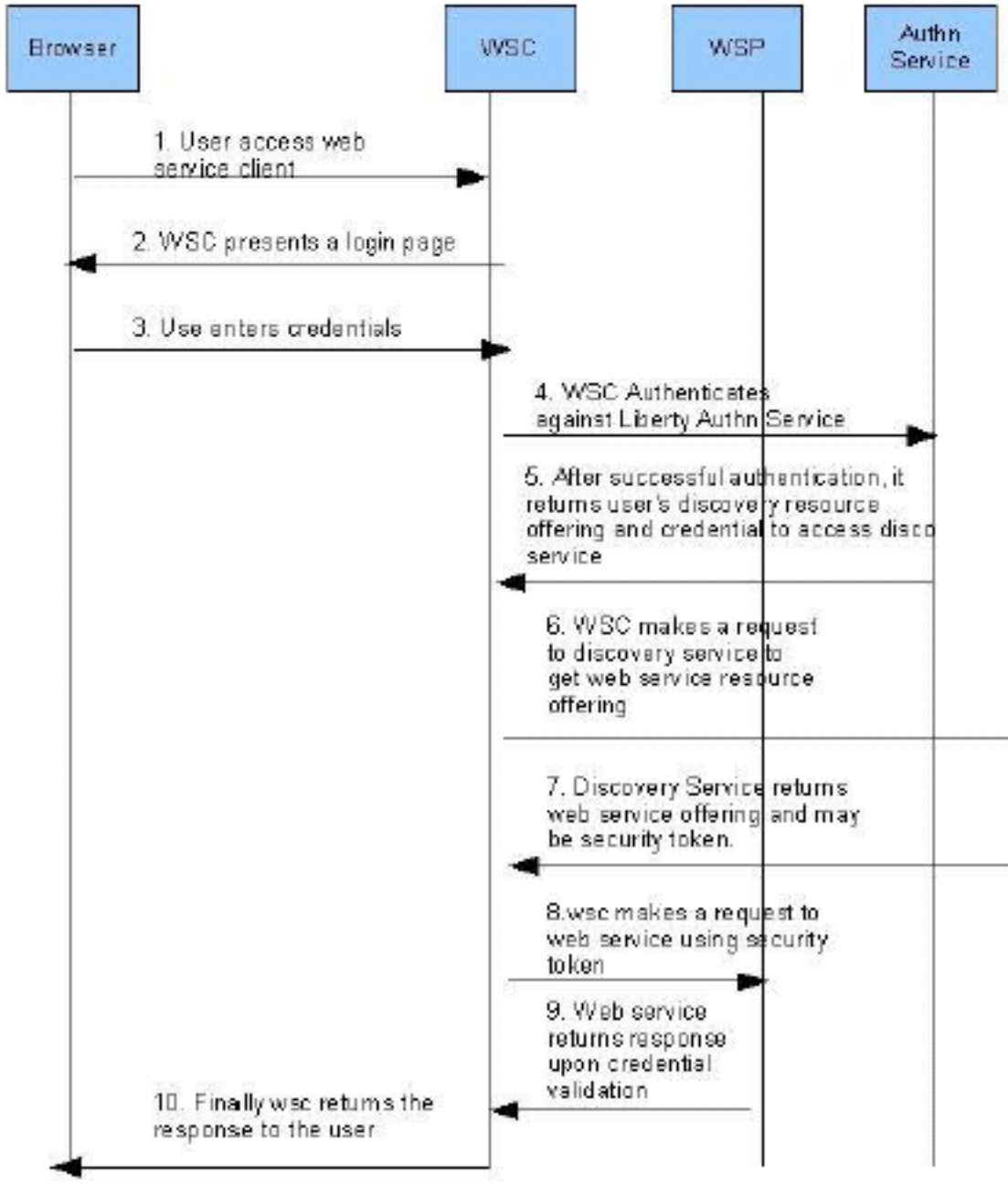


FIGURE 13–5 Web Services Stack Using Authentication Web Service

Implemented Services

Information on the services implemented in the Web Services Stack is in the following sections:

Additional information can be found in the *Sun Federated Access Manager 8.0 Administration Guide* and the *Sun Federated Access Manager 8.0 Developer's Guide*.

Authentication Web Service

The Authentication Web Service is for provider-to-provider authentication. SOAP defines an XML-based messaging paradigm, but not security mechanisms for message protection; particularly, they do not describe user authentication. To secure SOAP messages, the Authentication Web Service was implemented based on the Liberty ID-WSF Authentication Service and Single Sign-On Service Specification. The specification defines a protocol that adds the Simple Authentication and Security Layer (SASL) authentication functionality. Upon successful authentication, the final Simple Authentication and Security Layer (SASL) response contains the resource offering for the Discovery Service.

Note – The Authentication Web Service is configured using the XML service file `amAuthnSvc.xml` and can be managed using the OpenSSO Enterprise console or this XML file. Additional administration information can be found in the *Sun Federated Access Manager 8.0 Administration Guide*.

The following sections contain more general information.

Authentication Web Service Process

The exchange of authentication information between a web service consumer (WSC) and web service provider (WSP) is accomplished using SOAP-bound messages. The messages are a series of client requests and server responses specific to the defined SASL mechanism (or mode of authentication). The authentication exchange can involve an arbitrary number of round trips, dictated by the particular SASL mechanism employed. The WSC might have knowledge of the supported SASL mechanisms, or it might send the server its own list of mechanisms and allow the server to choose one. (The list of supported mechanisms can be found at SASL

Mechanisms.) After receiving a request for authentication (or any response from the WSC), the WSP may issue additional challenges or indicate authentication failure or success. The sequence between the WSC and the Authentication Web Service (a WSP) is as follows.

1. The authentication exchange begins when a WSC sends a SASL authentication request to the Authentication Web Service on behalf of a principal. The request message contains an identifier for the principal and indicates one or more SASL mechanisms from which the service can choose.
2. The Authentication Web Service responds by asserting the method to use and, if applicable, initiating a challenge.

Note – If the Authentication Web Service does not support any of the cited methods, it responds by aborting the exchange.

---

3. The WSC responds with the necessary credentials for the chosen method of authentication.
4. The Authentication Web Service replies by approving or denying the authentication. If approved, the response includes the credentials the WSC needs to invoke other web services, such as the Discovery Service.

Authentication Web Service API

The Authentication Web Service includes the following Java programming packages:

- `com.sun.identity.liberty.ws.authnsvc` is a client API for external Java applications to send SASL requests and receive SASL responses.
- `com.sun.identity.liberty.ws.authnsvc.mechanism` defines an interface to handle different SASL mechanisms.
- `com.sun.identity.liberty.ws.authnsvc.protocol` contains classes that represent the SASL request and response.

Together, the packages are used to initiate the authentication process and communicate authentication credentials to the Authentication Web Service. For more information, see the *Federated Access Manager 8.0 Java API Reference* and the *Sun Federated Access Manager 8.0 Developer's Guide*.

Which Authentication Service to Use?

The Authentication Web Service is not to be confused with the proprietary OpenSSO Enterprise Authentication Service. Architecturally, the Authentication Web Service sits on top of the OpenSSO Enterprise Authentication Service and the Web Services Stack framework. You might use the Authentication Web Service if you are a service provider and want to use a standards-based mechanism to authenticate users. Following are two use cases where the Authentication Web Service is preferable to the proprietary OpenSSO Enterprise Authentication Service:

- A service provider relies on a remote identity provider (not necessarily using OpenSSO Enterprise) for authentication.
- An enterprise in a service-oriented architecture (SOA) environment wants to use non-proprietary mechanisms to authenticate users and web services clients before accessing a protected web service.

In addition to providing an authentication service to verify credentials (for example, user ID and password), the Authentication Web Service provides the WSC with bootstrap information that contains the endpoint and credentials needed to access the Discovery Service. The WSC can ignore the bootstrap or use it to access other web services, such as the authenticated user's personal profile or calendar.

Note – An authenticated enterprise might also use the bootstrap information to access a partner in a business-to-business environment.

Following is a scenario that shows how the Authentication Web Service interacts with the OpenSSO Enterprise Authentication Service. The WSC delegates all authentication to the identity provider and prefers PLAIN authentication which accepts a user identifier and password.

1. The user attempts access to a service provider (not necessarily hosted by OpenSSO Enterprise).
2. When the service provider (acting as a WSC) finds that the user is not authenticated, it invokes the identity provider Authentication Web Service by sending a SOAP request.

   Note – It is assumed that the identity provider is hosted by OpenSSO Enterprise where the OpenSSO Enterprise Authentication Service is configured for Certificate and LDAP authentication and the Authentication Web Service has mapped LDAP to its own PLAIN authentication mechanism)

   ---

3. After inspecting its configuration, the Authentication Web Service sends back a response indicating that it supports Certificate and PLAIN authentication.
4. The service provider decides on PLAIN authentication and prompts the user for an identifier and password.
5. The service provider receives the user identifier and password and sends it to the identity provider.
6. The identity provider passes the credentials to the locally hosted LDAP Authentication module using the proprietary OpenSSO Enterprise Authentication Service Java API.
7. The LDAP Authentication module verifies the credentials by accessing the user's personal profile (hosted by a third-party product).

8. The Authentication Web Service is notified of the verification and sends a response to the service provider indicating successful authentication. If configured to do so, it also includes bootstrap information formatted using XML and containing the Discovery Service endpoint and a token to invoke it.
9. The service provider parses the response, verifies that it is a successful authentication, and provides the service to the user.

At some point the service provider might need to access the user's personal profile. To do this, it will use the bootstrap information received during this process to contact the Discovery Service and find where the profile is stored. The Discovery Service returns a *resource offering* (containing a token and the location of an endpoint), and the service provider uses that to invoke the Liberty Personal Profile Service.

Discovery Service

All web services are defined by a Web Services Description Language (WSDL) file that describes the type of data the service contains, the available ways said data can be exchanged, the operations that can be performed using the data, a protocol that can be used to perform the operations, and a URL (or endpoint) for access to the service. Additionally, the WSDL file itself is assigned a unique resource identifier (URI) that is used to locate it in a Universal Description, Discovery and Integration (UDDI) repository. Thus, the web service can now be discovered.

According to the Web Services Glossary, *discovery* of a web service is the act of locating a WSDL file for it. Typically, there are one or more web services on a network so, a discovery service is required to keep track of them. OpenSSO Enterprise implements the *Liberty ID-WSF Discovery Service Specification* for its Discovery Service. When a WSC queries the Discovery Service for a WSP that allows access to a particular user's credit card information, the Discovery Service matches the properties in the request against the properties of its registered services and returns the appropriate *resource offering* (which defines an association between a type of identity data and a URI to the WSDL definition for obtaining access to the data.)

Note – The Discovery Service is configured using the XML service file `amDisco.xml` and can be managed using the OpenSSO Enterprise console or this XML file. Additional administration information can be found in the *Sun Federated Access Manager 8.0 Administration Guide*.

The following sections contain additional information on the Discovery Service.

- "Discovery Service Process" on page 195
- "Discovery Service Architecture" on page 196
- "Discovery Service API" on page 197

Discovery Service Process

Remark 13–2 Reviewer **Is it correct to say that this process is not generating security tokens? Also, please verify: a few changes were made to this section.**

Figure 13–6 provides an overview of the interactions between the parties in a web services environment using the Discovery Service. In this scenario, the identity provider hosts the Discovery Service and the process assumes that the Discovery Service is not generating security tokens.

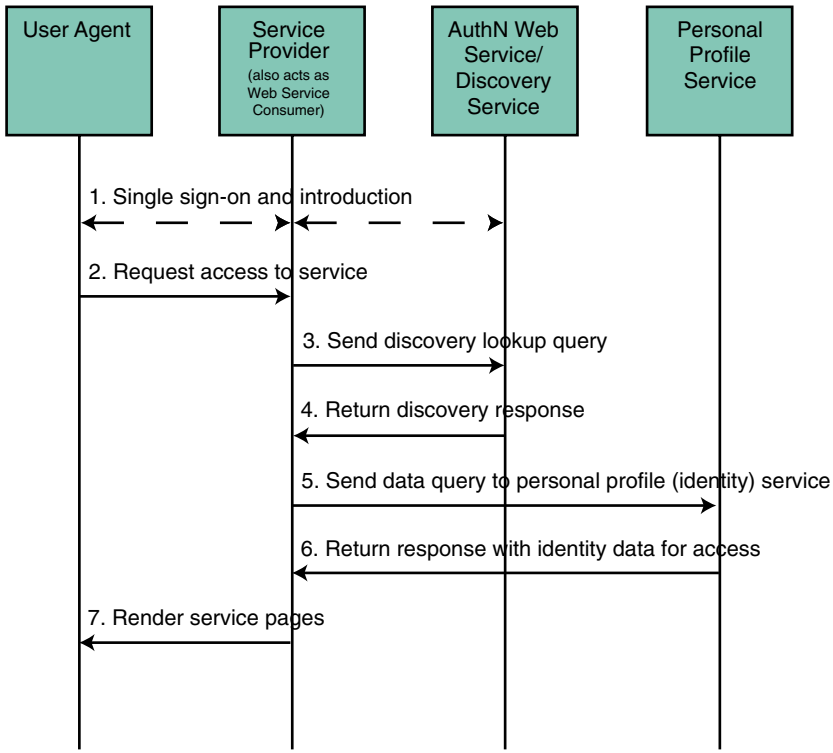


FIGURE 13–6 Discovery Service Process

1. The user logs in to an identity provider, is authenticated, and completes the federation process, enabling single sign-on with other members of the circle of trust. More specifically:
   a. Within a browser, the user types the URL for a service provider.
   b. The service provider collects the user's credentials and redirects the information to the identity provider for authentication.
   c. If the credentials are verified, the user is authenticated.

d. Assuming the identity provider is the center of a circle of trust, it will notify the authenticated user of the option to federate any local identities created with circle of trust member organizations. The user would then accept or decline this invitation to federate. If the user accepts this option to federate, single sign-on is enabled. By accepting the invitation, the user will be given the option to federate to a member organization's web site at each login.

2. After authentication, the user now requests access to services hosted by another service provider in the circle of trust.
3. The service provider, acting as a WSC, sends a `DiscoveryLookup` query to the Discovery Service looking for a pointer to the user's identity provider.

   The service provider is able to bootstrap the Discovery Service using the end point reference culled from the authentication statement.
4. The Discovery Service returns a `DiscoveryLookup` response to the service provider that points to the instance of the requested identity provider.

   The response contains the resource offering for the user's Liberty Personal Profile Service.
5. The service provider then sends a query (using the XXXXXX Data Services Template Specification) to the Liberty Personal Profile Service.

   The required authentication mechanism specified in the Liberty Personal Profile Service resource offering must be followed.
6. The Liberty Personal Profile Service authenticates and validates authorization, or policy, or both for the requested user and service provider, and returns a Data Services Template response.

   If user interaction is required for some attributes, the Interaction Service will be invoked to query the user for consents or attribute values. The Data Services Template would then be returned after all required data is collected.
7. The service provider processes the Liberty Personal Profile Service response and renders HTML pages based on the original request and user authorization.

   A user's actual account information is not exchanged during federation. Thus, the identifier displayed on each provider site will be based on the respective local identity profile.

Discovery Service Architecture

Remote Java applications use the Client SDK to form requests sent to the Discovery Service and to parse the responses received back from it. Requests are initially received by the `SOAPReceiver`, a servlet which constructs the SOAP message that incorporates the client request.

Note – The SOAP Binding Service defines how to send and receive messages using SOAP, an XML-based messaging protocol. For more information, see XXXXXX Chapter 9, SOAP Binding Service.

The SOAP message is then routed to the Discovery Service which parses the resource identifier from it. This identifier is used to find a matching user distinguished name (DN). The necessary information is then culled from the corresponding profile, a response is generated, and returned to the `SOAPReceiver` which sends the response back to the client. Figure 13–7 illustrates this architecture.

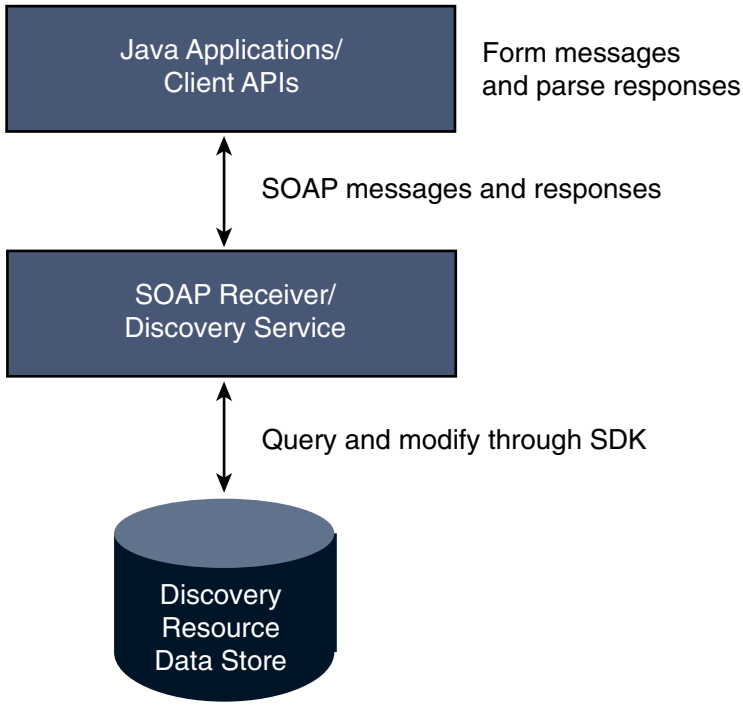


FIGURE 13–7 Discovery Service Architecture

Discovery Service API

The Discovery Service includes the following Java programming packages:

- `com.sun.identity.liberty.ws.disco` is a client API that provides interfaces to communicate with the Discovery Service.

- `com.sun.identity.liberty.ws.disco.plugins` defines an interface that can be used to develop plug-ins. The package also contains some default plug-ins.
- `com.sun.identity.liberty.ws.interfaces` contains interfaces that can be used to implement functionality common to all Liberty-enabled identity services. Several implementations of these interfaces have been developed for the Discovery Service.

For more information, see the *Federated Access Manager 8.0 Java API Reference* and the *Sun Federated Access Manager 8.0 Developer's Guide*.

SOAP Binding Service

The Web Services Stack uses SOAP messages to convey identity data between providers. OpenSSO Enterprise has implemented the Liberty ID-WSF SOAP Binding Specification (Liberty ID-WSF-SBS) as the method of transport for this purpose. The specification defines SOAP as the binding to HTTP, which is itself layered onto the TCP/IP stack. The SOAP Binding Service is a set of Java API used by the developer to send and receive SOAP messages.

Note – The SOAP Binding Service is configured using the XML service file `amSOAPBinding.xml` and can be managed using the OpenSSO Enterprise console or this XML file. Additional administration information can be found in the *Sun Federated Access Manager 8.0 Administration Guide*.

The following sections contain additional information on the SOAP Binding Service.

- "SOAP Binding Service Components" on page 198
- "SOAP Binding Service Process" on page 199
- "Discovery Service API" on page 197

SOAP Binding Service Components

The following sections contain information on some programming components of the SOAP Binding Service.

- "`SOAPReceiver` Servlet" on page 198
- "`RequestHandler` Interface" on page 199

`SOAPReceiver` Servlet

The `SOAPReceiver` servlet receives a `Message` object from a WSC, verifies the signature, and constructs its own `Message` object for processing by OpenSSO Enterprise. The `SOAPReceiver` then invokes the correct request handler class to pass this second `Message` object on to the appropriate OpenSSO Enterprise service for a response. When the response is generated, the `SOAPReceiver` returns this `Message` object back to the WSC.

`RequestHandler` Interface

`com.sun.identity.liberty.ws.soapbinding.RequestHandler` is an interface that must be implemented on the server side by any Liberty-based web service using the SOAP Binding Service. For more information, see the *Federated Access Manager 8.0 Java API Reference* and the *Sun Federated Access Manager 8.0 Developer's Guide*.

SOAP Binding Service Process

In the SOAP Binding Service process, an identity service invokes the `Message` class (contained in the Client SDK) to construct a request. (As clients of the SOAP Binding Service, the Discovery Service, the Liberty Personal Profile Service (and the sample Employee Profile Service), and the Authentication Web Service all use the SOAP Binding Service client-side API.) The `Message` object will contain any default or non-default SOAP headers as well as the SOAP body containing the request(s). Once generated, the WSC invokes the `sendRequest` method and sends the `Message` object to the `SOAPReceiver` which receives the `Message`, verifies the signature, and constructs its own `Message` object. The `SOAPReceiver` then invokes the appropriate Request Handler class to send this second message to the corresponding service for a response.

The web service processes the second message, generates a response, and sends that response back to the `SOAPReceiver` which, in turn, returns the response back to the WSC for processing.

Note – Before invoking a corresponding service, the SOAP framework might also do the following:

- Authenticate the sender identity to verify the credentials of a WSC peer, probably by verifying its client certificate.
- Authenticate the invoking identity to verify the credentials of a WSC on behalf of a user to verify whether the user has been authenticated. This depends on the security authentication profile.
- Granular authorization to authorize the WSC before processing a service request.

SOAP Binding Service API

The SOAP Binding Service includes a Java package named `com.sun.identity.liberty.ws.soapbinding`. This package provides classes to construct SOAP requests and responses and to change the contact point for the SOAP binding. For more information, see the *Federated Access Manager 8.0 Java API Reference* and the *Sun Federated Access Manager 8.0 Developer's Guide*.

Liberty Personal Profile Service

A *data service* is a web service that supports the query and modification of data regarding a principal. An example of a data service is a web service that hosts and exposes a principal's profile information, such as name, address and phone number. A *query* is when a WSC requests and receives the data (in XML format). A *modify* is when a WSC sends new information to update the data. The Liberty Alliance Project has defined the *Liberty ID-WSF Data Services Template Specification* (Liberty ID-WSF-DST) as the standard protocol for the query and modification of identity profiles exposed by a data service. The *Liberty ID-Service Interface Specification Personal Profile Service Specification* (Liberty ID-SIS-PP) describes a data service that provides an identity's basic profile information, such as full name, contact details, and financial information. This data service is intended to be the least common denominator for holding consumer-based information about a principal. OpenSSO Enterprise has implemented these specifications and developed the Liberty Personal Profile Service which can be queried for identity data and its attributes can be updated.

Note – The Liberty Personal Profile Service is configured using the XML service file `amLibertyPersonalProfile.xml` and can be managed using the OpenSSO Enterprise console or this XML file. Additional administration information can be found in the *Sun Federated Access Manager 8.0 Administration Guide*.

The following sections contain additional information on the Liberty Personal Profile Service.

Liberty Personal Profile Service Design

The Liberty ID-WSF-DST specifies a base layer that can be extended by any instance of a data service. An example of a data service is an identity service, such as an online corporate directory. When you want to contact a colleague, you conduct a search based on the individual's name, and the data service returns information associated with that person's identity. The information might include the individual's office location and phone number, as well as job title or department name. For proper implementation, all data services must be built on top of the Liberty ID-WSF-DST because it provides the data model and message interfaces. Figure 13–8 illustrates how OpenSSO Enterprise uses the Liberty ID-WSF-DST as the framework for the Liberty Personal Profile Service and other custom data services.

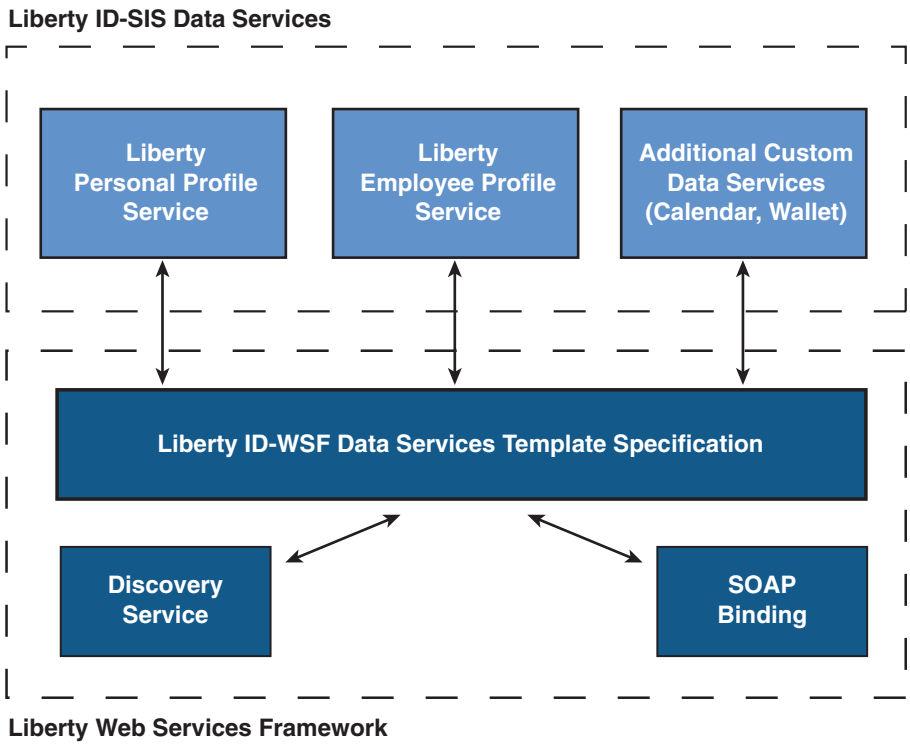


FIGURE 13–8 Data Service Template as Building Block of Data Services

Note – For more information on the data services specification, see the *Liberty ID-WSF Data Services Template Specification*. For more information on the personal profile specifications, see the *Liberty ID-SIS Personal Profile Service Specification*.

Liberty Personal Profile Service Process

The invocation of a personal profile begins when a WSC posts a query or a modify request to the Liberty Personal Profile Service on behalf of a user. Figure 13–9 illustrates the invocation process of the Liberty Personal Profile Service.

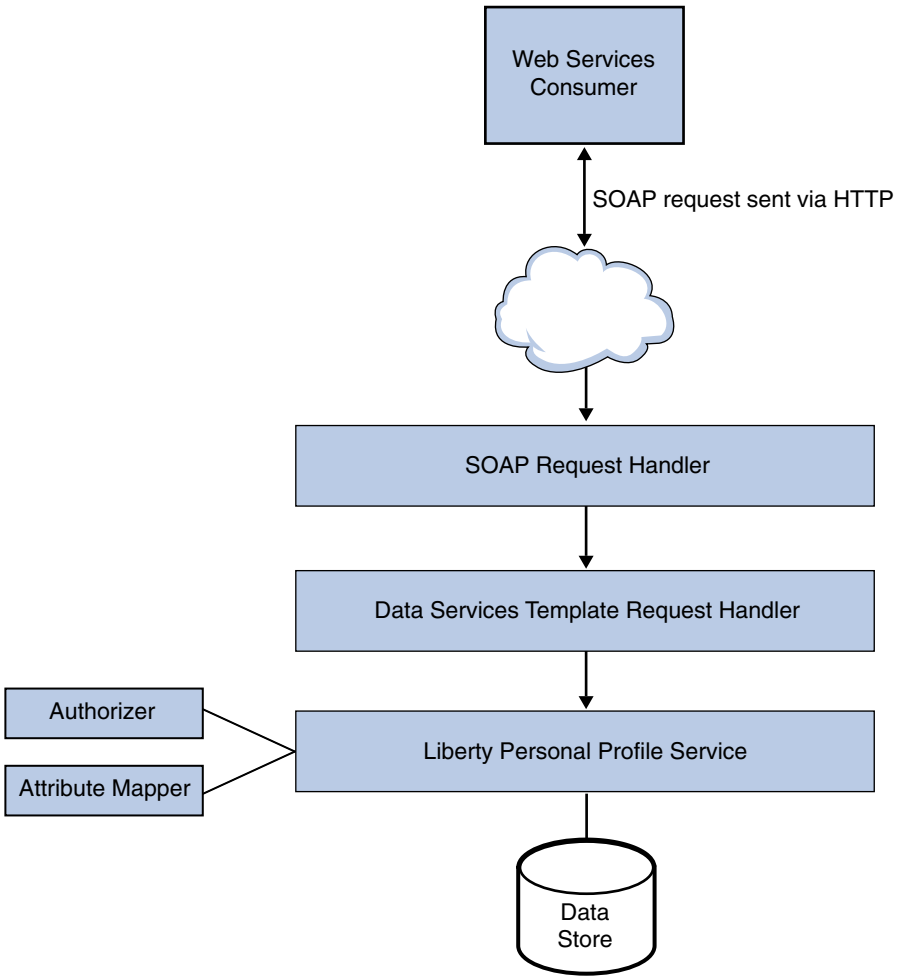


FIGURE 13–9 Liberty Personal Profile Service Process

1. A WSC uses the Data Services Template API uses SOAP to post a query or a modify request to the Liberty Personal Profile Service.
2. The SOAP request is received by the `SOAPReceiver` servlet provided by the SOAP Binding Service.

   The `SOAPReceiver` invokes either the Discovery Service, the Authentication Web Service, or the Liberty Personal Profile Service, depending on the service key transmitted as part of the URL. The SOAP Binding Service might also authenticate the client identity. For more information, see "`SOAPReceiver` Servlet" on page 198.
3. The Liberty Personal Profile Service implements the `DSTRequestHandler` to process the request.

The request is processed based on the type (query or modify) and the query expression. Processing might entail the authorization of a WSC using the OpenSSO Enterprise Policy Service, or it might entail using the Interaction Service for interacting with the user before sending data to the WSC.

4. The Liberty Personal Profile Service builds a service response, adds credentials (if they are required), and sends the response back to the WSC.
   - For a response to a query request, the Liberty Personal Profile Service builds a personal profile container (as defined by the specification). It is formatted in XML and based on the Query Select expression. The Liberty Personal Profile Service attribute values are extracted from the data store by making use of the attribute mapper. The attribute mapper is defined by the XML service file, and the attribute values will be used while building the XML container. The Liberty Personal Profile Service then applies `xpath` queries on the XML and provides us with the resultant XML data node.
   - For a response to a modify request, the Liberty Personal Profile Service parses the Modifiable Select expression and updates the new data from the new data node in the request.

Note – For initial access, the hosting provider of the Liberty Personal Profile Service needs to be registered with the Discovery Service on behalf of each identity principal. To register a service with the Discovery Service, update the resource offering for that service. For more information, see the *Sun Federated Access Manager 8.0 Administration Guide*.

Data Services API

OpenSSO Enterprise data services are built using a Java package called `com.sun.identity.liberty.ws.dst`. OpenSSO Enterprise provides this package for developing custom services based on the Liberty ID-WSF-DST. Additional information about these interfaces can be found in XXXXX Data Services Template API and in the *Federated Access Manager 8.0 Java API Reference*.

OpenSSO Enterprise contains two packages based on the Liberty ID-WSF-DST. They are: * com.sun.identity.liberty.ws.dst Package * com.sun.identity.liberty.ws.dst.service Package

- `com.sun.identity.liberty.ws.dst` provides classes for the Client SDK.
- `com.sun.identity.liberty.ws.dst.service` provides a handler class that can be used by any data service that is built using the Liberty ID-SIS Specifications. The `DSTRequestHandler` class is used to process query or modify requests sent to an identity data service. It is an implementation of the interface `com.sun.identity.liberty.ws.soapbinding.RequestHandler`.

For more information, see the *Federated Access Manager 8.0 Java API Reference* and the *Sun Federated Access Manager 8.0 Developer's Guide*.

◆ ◆ ◆ CHAPTER 14

Delivering Identity Web Services

OpenSSO Enterprise provides client interfaces for authentication, authorization, session, identity management and auditing in Java, in C (C++) and in HTTP(S)/XML. These interfaces are used by web and Java EE policy agents as well as custom applications developed externally. Now, OpenSSO Enterprise also delivers web services that expose these identity functions as simple web services. This chapter contains information on the following topics:

About Identity Web Services

A *web service* is a black-box component that can be accessed using exposed endpoints. OpenSSO Enterprise uses this concept to expose the following security and identity related functions as Identity Web Services:

- authenticate (user credential verification)
- authorize (an authenticated identity's access permission)
- attributes (an authenticated identity's profile)
- log (record and audit actions)

Identity Web Services allow developers to easily invoke these functions without any knowledge of OpenSSO Enterprise, resolving the problems of enabling web service discovery and invocation, security, privacy and ease-of-deployment. Keeping Identity Web Services simple allows an application developer to consume them by pointing an integrated development environment (IDE) to the service's URL and allowing it to generate the stub code that wraps a call to the service. Identity Web Services are supported on:

- NetBeans
- Eclipse
- Visual Studio

Note – Identity Web Services does not require the Client SDK or deployment of an agent or proxy to protect a resource.

Within Identity Web Services the user enters authentication credentials using a JavaServer Pages (JSP). The user data is then forwarded to the composite application which authenticates the web service request. The application may then authorize the operation and obtain the user's profile. See "Identity Web Service Styles" on page 206 for more information.

Identity Web Service Styles

OpenSSO Enterprise Identity Web Services have been developed in two styles. The decision on which style to use is the initial choice when designing your application. The styles are:

- The SOAP and Web Services Description Language (WSDL) style is the traditional approach preferred by the service-oriented architecture (SOA) business intelligence community. See "SOAP and WSDL" on page 206.
- The REpresentational State Transfer (REST) style is a newer approach preferred by the Web 2.0 community. (A REST service is referred to as *RESTful*.) See "REST" on page 207.

Note – `developers.sun.com` has an excellent three part article called Securing Applications with Identity Services which contains IDE configuration information and procedures.

SOAP and WSDL

SOAP, WSDL, and XML Schema have become the standard for exchanging XML-based messages among applications. To implement this style, the IDE must obtain the WSDL, generate the client stubs, and set up the JavaServer Pages (JSP) for the Identity Web Services. Once completed, the SOAP Identity Web Services are accessible with the following URLs:

- **`http://`***host_machine.domain***`:8080/opensso/identityservices/IdentityServices`**
- **`http://`***host_machine.domain***`:8080/opensso/identityservices?WSDL`**

This style may be appropriate when:

- A formal contract must be established to describe the interface that the web service offers. A WSDL is needed to describe the web service interfaces including details such as messages, operations, bindings, and location.
- The architecture must address complex requirements including security, financial transactions, provider trust and the like.

- The architecture needs to handle asynchronous processing and invocation. The infrastructure provided by standards such as WSRM and APIs such as JAX-WS can be leveraged out of the box.

REST

The internet is comprised of *resources*. Clients may access resources with a URL. When requested, a representation of the resource (an HTML page) is returned. The result of the user clicking a link on the page is that another resource is accessed (possibly an image, video, or another HTML page). Each new representation places the client into a state that is different from the previous state. Thus, the client application changes state with each accessed resource representation. REST is a design architecture in which a web service is viewed as a resource identified by a URL. The web service client then accesses it using a globally defined set of remote methods that describe the action to be performed. REST is not a standard; you can only understand it, and design web services in the REST style. REST does, though, use standards including:

- HTTP
- URLs
- Resource representations (XML, HTML, GIF, JPEG, and others)
- MIME types (text/xml, text/html, image/gif, image/jpeg, and others)

RESTful services are accessed using a generic interface; in OpenSSO Enterprise it is the GET, POST, PUT, and DELETE HTTP methods. The RESTful Identity Web Service is accessible at **`http://`***host_machine.domain*`:8080/opensso/identity`. Because these web services are exposed using the HTTP methods, they can be accessed from a browser. This style may be appropriate when:

- The web services are completely stateless. A good test is to consider whether the interaction can survive a restart of the server.
- Bandwidth needs to be limited. REST is particularly useful for limited-profile devices such as PDAs and mobile phones, where the XML payload must be restricted.
- Aggregation into existing web sites is needed. Web services can be exposed with XML and consumed by HTML without significantly reinventing the existing web site architecture.

Note – OpenSSO Enterprise REST interfaces currently support only username and password authentication.

Identity Web Services Architecture

In an Identity Web Service interaction, the user interacts with an application which then calls either of the Identity Web Services to authenticate and authorize the identity, create personalized services, and log the actions. When contacted at the respective URL, OpenSSO Enterprise obtains the user profile from the appropriate identity repository for authentication and the policy configuration from the appropriate configuration data store, and writes the actions to the configured log file. Figure 14–1 illustrates the components of the Identity Web Services.

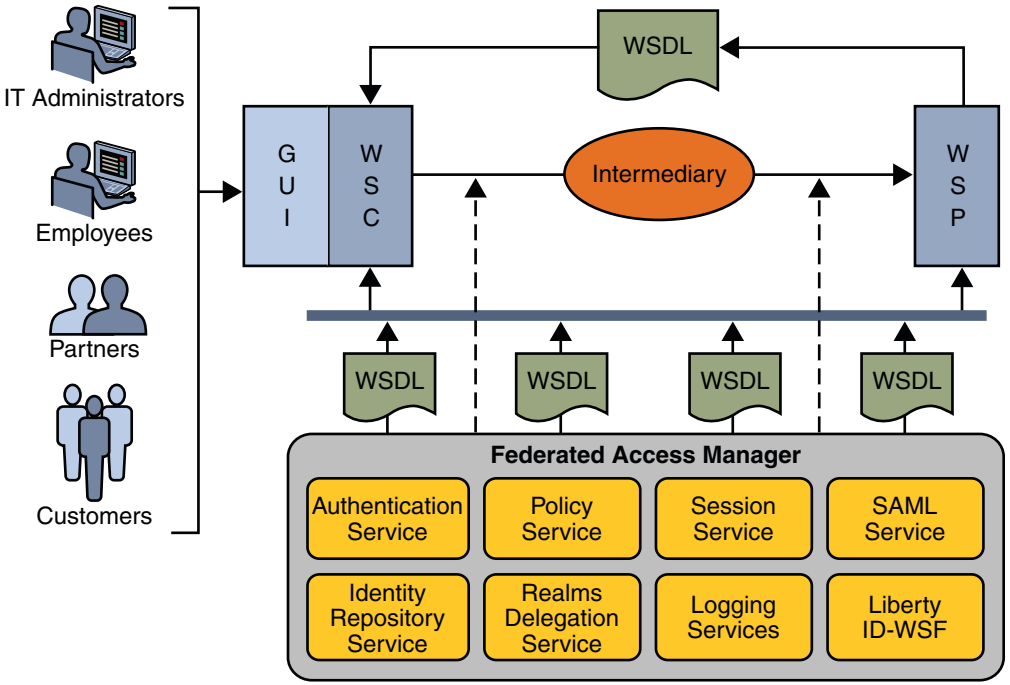


FIGURE 14–1 Components within the Identity Services Interactions

◆ ◆ ◆ CHAPTER 15

Securing Web Services

OpenSSO Enterprise implements security for web services. This chapter contains the following sections concerning this functionality.

About Web Services Security

A web service exposes some type of functionality using a platform-independent interface. Enterprises use platform-independent interfaces as a mechanism for allowing their applications to cross network boundaries and communicate with those of their partners, customers and suppliers. Web services are accessed by sending a request to one of the service's defined endpoints but the built-in openness of the web services technologies creates security risks. The following security requirements have been identified and must be supported to insure that the communications between a web service provider (WSP) and a web service client (WSC) are not compromised.

- Data integrity and confidentiality during transport
- Authentication of the sending entity
- Message uniqueness

Initially, securing web services communications was addressed on the transport level, relying on securing the HTTP transmissions themselves using Secure Sockets Layer (SSL). This is not adequate though when access to an application is requested through an intermediary. The solution to this is to encrypt the entire request using *message level security* before it leaves the WSC. In message level security, authentication and authorization information in the form of a token is contained within the SOAP header, allowing the request or response to securely pass through multiple intermediaries before reaching its intended receiver. Figure 15–1 depicts message level security.

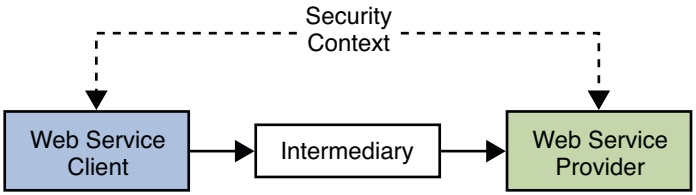


FIGURE 15–1 Message Level Security

Note – Additionally, policy enforcement points are distributed throughout the environment and access to resources and services is mainly over HTTP.

The OpenSSO Enterprise Web Services Security functionality can be used in a variety of platforms and containers for different purposes. These can range from providing single sign-on and federation support for web applications to securing web applications using security agents deployed on the appropriate web container. Here are the web services security functions that OpenSSO Enterprise provides.

- Identify and authenticate the principal.
- Generate and exchange security tokens in a trusted environment.
- Preserve the identity through multiple intermediaries and across domain boundaries.
- Maintain privacy and integrity of identity data.
- Log the outcome.

Basically, the JCP, W3C, and OASIS are developing specifications related to web services security. WS-I creates profiles that recommend what to implement from various specifications and provides direction on how to implement the specifications. The Liberty Alliance provides a framework for building interoperable identity services, using specifications such as WS-Security (OASIS), SOAP, Security Assertions Markup Language SAML (OASIS) and XML (W3C). The following sections briefly discuss the specifications and profiles being developed by each organization.

There are a number of web services security specifications, guidelines, and tools that have been implemented to develop these Web Services Security features for OpenSSO Enterprise. The following sections have more general information on these specifications.

- "Web Services Interoperability Technology" on page 211
- "WS-Security Specification" on page 211
- "WS-Trust Specification" on page 212
- "Liberty Alliance Project Specifications" on page 212
- "JSR-196 Specification" on page 212

Web Services Interoperability Technology

Web Services Interoperability Technology (WSIT) is an open source implementation of many of the web services specifications (commonly referred to as **WS-***). The project was started by Sun Microsystems, and consists of Java API that allow developers to create web service clients and services that enables operations between the Java platform and clients and servers developed with the WS-* specifications. WSIT provides implementation of the following specifications for interoperability with .NET 3.0.

- WS-Metadata Exchange
- WS-Transfer
- WS-Reliable Messaging
- WS-Reliable Messaging Policy
- WS-Atomic Transaction
- WS-Coordination
- WS-Security 1.0 and 1.1
- WS-Security Policy
- WS-Trust
- WS-Secure Conversation
- WS-Policy
- WS-Policy Attachment

Note – Web service specifications are referred to collectively as **WS-*** although there is neither a single managed set of specifications that this consistently refers to nor one recognized body that owns all the specifications. **WS-*** is a general nod to the fact that many specifications use WS as their prefix.

Sun is working closely with Microsoft to ensure interoperability of web services enterprise technologies such as message optimization, reliable messaging, and security. The initial release of the Web Services Interoperability Technologies (WSIT) is a product of this joint effort. WSIT is an implementation of a number of open web services specifications to support enterprise features such as message optimization, reliable messaging, security, bootstrapping and configuration. More information can be found in this WSIT tutorial on java.sun.com.

WS-Security Specification

The WS-Security specification is now developed by the Organization for Advancement of Structured Information Standards (OASIS) after being submitted to the standards body by IBM, Microsoft, and VeriSign in 2002. The specification defines how to sign a SOAP message and describes enhancements that provide message integrity, message confidentiality, and message authentication. It also defines:

- An extensible, general-purpose mechanism for associating security tokens with message content.

- How to encode binary security tokens.
- A framework for XML-based tokens.
- How to include opaque encrypted keys.

The WS-Security specification is designed to be used together with other WS-* specifications to provide tools for secure and reliable web services transactions. For example, WS-Policy does not provide a solution for negotiating web services in and of itself; it is used in conjunction with other specifications to accommodate a wide variety of policy exchange models. WS-Trust, on the other hand, uses the secure messaging mechanisms of WS-Security to define extensions for security token exchange within different trust domains. For more information, see the Web Services Security page on the OASIS web site.

WS-Trust Specification

Web Services Trust Language (WS-Trust) uses the secure messaging mechanisms of WS-Security to define additional extensions for security token exchange and to enable the issuance and dissemination of credentials within different trust domains. WS-Trust is sued to develop the Security Token Service. For more information, see "Security Token Service" on page 215.

Liberty Alliance Project Specifications

The Liberty Alliance Project establishes open technical specifications that support a broad range of web services, driving the specifications for services such as Personal Profile Service and the Employee Profile Service. In order to build these identity web services, the Liberty Alliance Project provides a framework for identity federation and a framework for adjunct web services such as a registration service and a discovery service. For more general information on the Liberty Alliance Project, see "Using the Liberty ID-FF" on page 164 and the Liberty Alliance Project specifications.

JSR-196 Specification

The Java Community Process (JCP) primarily guides the development and approval of Java technical specifications, one of which is the Java Specification Request (JSR) 196. JSR 196 is a draft of the *Java Authentication Service Provider Interface for Containers* that defines a standard service provider interface (SPI) with which a message level *security agent* can be developed for Java EE containers on either the client side or the server side.

- A server side agent can be used to verify security tokens or signatures on incoming requests and extract principal data or assertions before adding them to the client security context.
- A client side agent can be used to add security tokens to outgoing requests, sign messages, and interact with the trusted authority to locate targeted web service providers.

The JSR–196 SPI is structured so that the security processes can be delegated to an agent at any of four interaction points (that represent the methods of the corresponding `ClientAuthModule` and `ServerAuthModule` SPI). These point are illustrated in Figure 15–2.

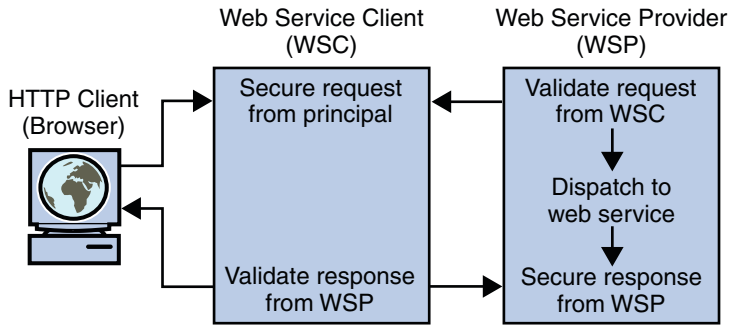


FIGURE 15–2 Four Security Process Points

When a WSC and WSP are both deployed in a Java EE web container protected by JSR–196 security agents, the initial request from the WSC is intercepted by the agent on the client side which then queries a *trusted authority* (for example, the Discovery Service) to retrieve the necessary authorization credentials to secure to the request. The secured request is then passed to the WSP. The agent on the provider side receives the request to validate the authorization credentials. If validation is successful, the request is exposed to the web service and a response is created using the sender's credentials and the application specific request. The response is then intercepted by the agent on the provider side to secure and return it to the WSC. Upon receiving the response, the agent on the client side validates it and dispatches it to the client browser. The JSR 196 draft specification is available at `http://www.jcp.org/en/jsr/detail?id=196`.

Web Services Security Architecture

The architectural strategy behind the Web Services Security framework is to model security agents on authentication and authorization SPI provided by the web container and to use a WSIT infrastructure for WS-Trust, WS-Policy and WS-I BSP security token implementations. Security agents secure web service requests and validate web service responses by inserting (or extracting) security tokens into (or out of) SOAP messages at the WSC and the WSP. This abstracts security from the application and allows customers to standardize security across multiple containers. Figure 15–3 illustrates this.

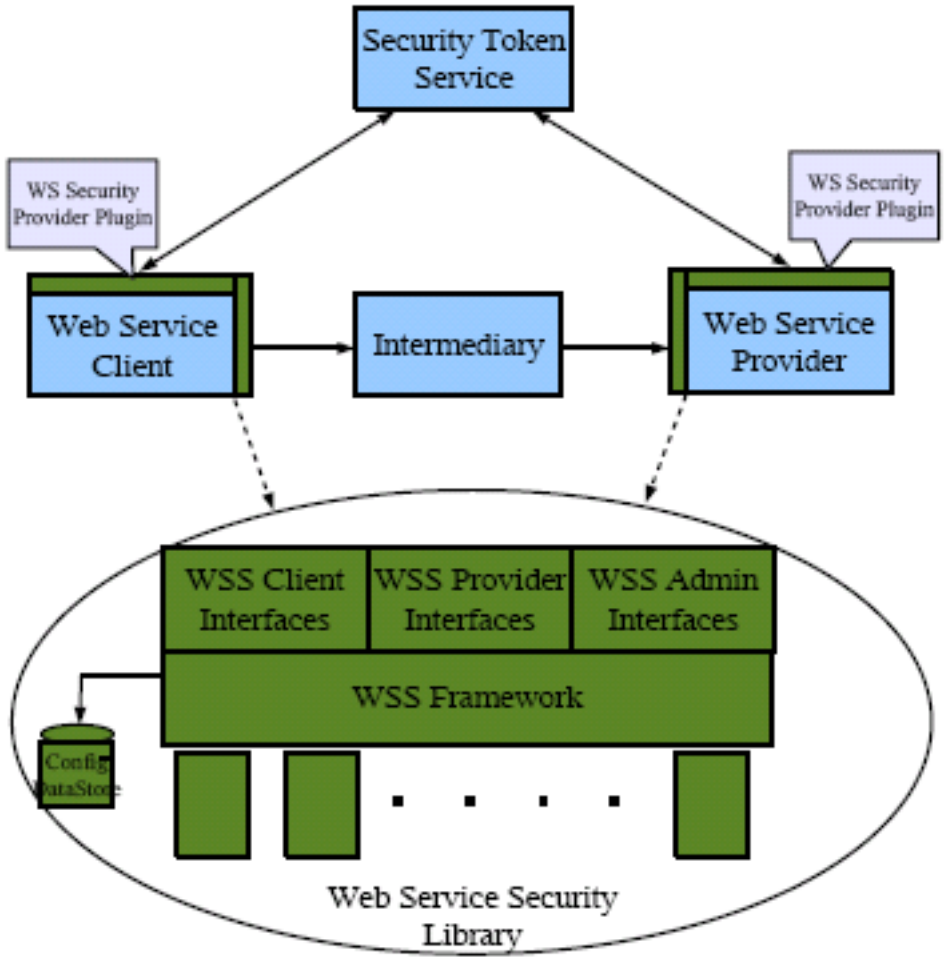


FIGURE 15–3 Architecture of Web Services Security Components

For an example, let's assume different domains with a WSC in domain 1 that wants to invoke a web service deployed at the WSP in domain 2. The WSC invokes the web service, using the service's Web Services Definition Language (WSDL) information to discern that a token is needed from the Security Token Service at domain B. The WSC invokes the Security Token Service in domain 2, using the service's WSDL information to discern that you can present a token from the Security Token Service at domain A. The WSC submits an X.509 signed request, using the WS-Trust protocol, for a SAML token from the Security Token Service at domain 1. Following this, the WSC submits the SAML token from domain 1 to the Security Token Service at domain 2 and receives back a SAML token from domain 2. The WSC now invokes the web

service at the WSP using the SAML token from domain 2. In this scenario, there is an implicit trust relationship between the two Security Token Services and the two domains.

Note – The WSC and the WSP invoke the Security Token Services using the Client SDK and OpenSSO Enterprise API. For better performance, over-the-wire calls can be eliminated by including token generation, conversion and validation SPI into the Client SDK.

The Web Services Security framework supports the following tokens.

- Tokens that can be authenticated:
  1. UserName
  2. X509
  3. SAML 1.1
  4. SAML 2.0
  5. Kerberos
- Tokens that can be issued:
  - UserName (generated with the Security Token Service or locally at the WSC)
  - X509 (generated with the Security Token Service or locally at the WSC)
  - SAML 1.1 (generated with the Security Token Service or locally at the WSC)
  - SAML 2.0 (generated with the Security Token Service or locally at the WSC)
  - Kerberos (generated locally at the WSC)

Web Services Security Components

In general, securing web services involves establishing trust between a WSC and a WSP. Towards this end, OpenSSO Enterprise provides security agents to verify (and extract data from) security tokens on incoming requests and to add security information (tokens and signatures) to outgoing responses. It also provides a Security Token Service to handle security tokens, and a number of Java interfaces. The following sections contain more information:

- "Security Token Service" on page 215
- "Security Agents" on page 219
- "Web Services Security Interfaces" on page 223

Security Token Service

Because of the use of tokens in Web Services Security, there is a need for a centralized token service; the Security Token Service serves this purpose for OpenSSO Enterprise. The Security Token Service was developed from the WS-Trust protocol which defines extensions to the WS-Security specification for issuing and exchanging security tokens and establishing and

accessing the presence of trust relationships. The Security Token Service is hosted as a servlet endpoint and coordinates security based interactions between a WSC and a WSP. The Security Token Service:

- Issues, renews, cancels, and validates security tokens.
- Allows customers to write their own plug-ins for different token implementations and for different token validations.
- Provides a WS—Trust based API for client and application access.
- Provides security tokens including Kerberos, Web Services-Interoperability Basic Service Profile (WS-I BSP), and Resource Access Control Facility (RACF).

When a WSC makes a call to the WSP, it first connects with the Security Token Service to determine the security mechanism and optionally obtain the security tokens expected by the WSP. (Alternately, the WSP could register its acceptable security mechanisms with the Security Token Service and, before validating the incoming SOAP request, could check with the Security Token Service to determine its security mechanisms.) Figure 15–4 illustrates a deployment architecture and the interactions between a WSC and a WSP using the Security Token Service.

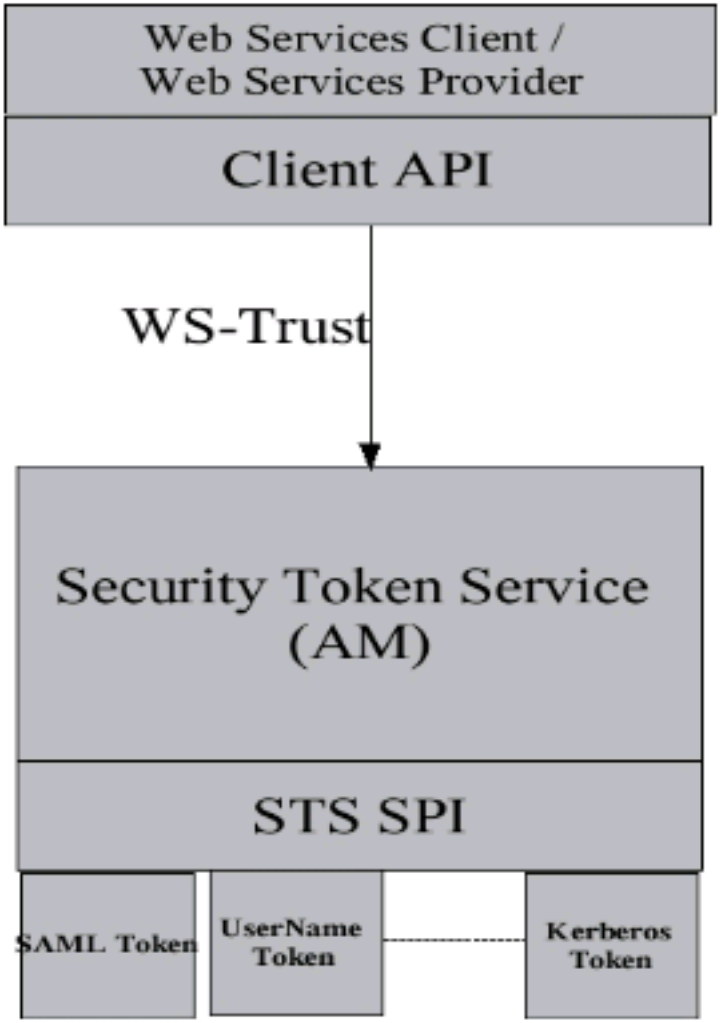


FIGURE 15–4 Security Token Service Architecture

When an authenticated WSC (carrying credentials that confirm either the identity of the end user or the application) requests a token for access to a WSP, the Security Token Service verifies the credentials and, in response, issues a security token that provides proof that the WSC has been authenticated. The WSC presents the security token to the WSP which verifies that the token was issued by a trusted Security Token Service. Figure 15–5 illustrates the design of the Security Token Service.

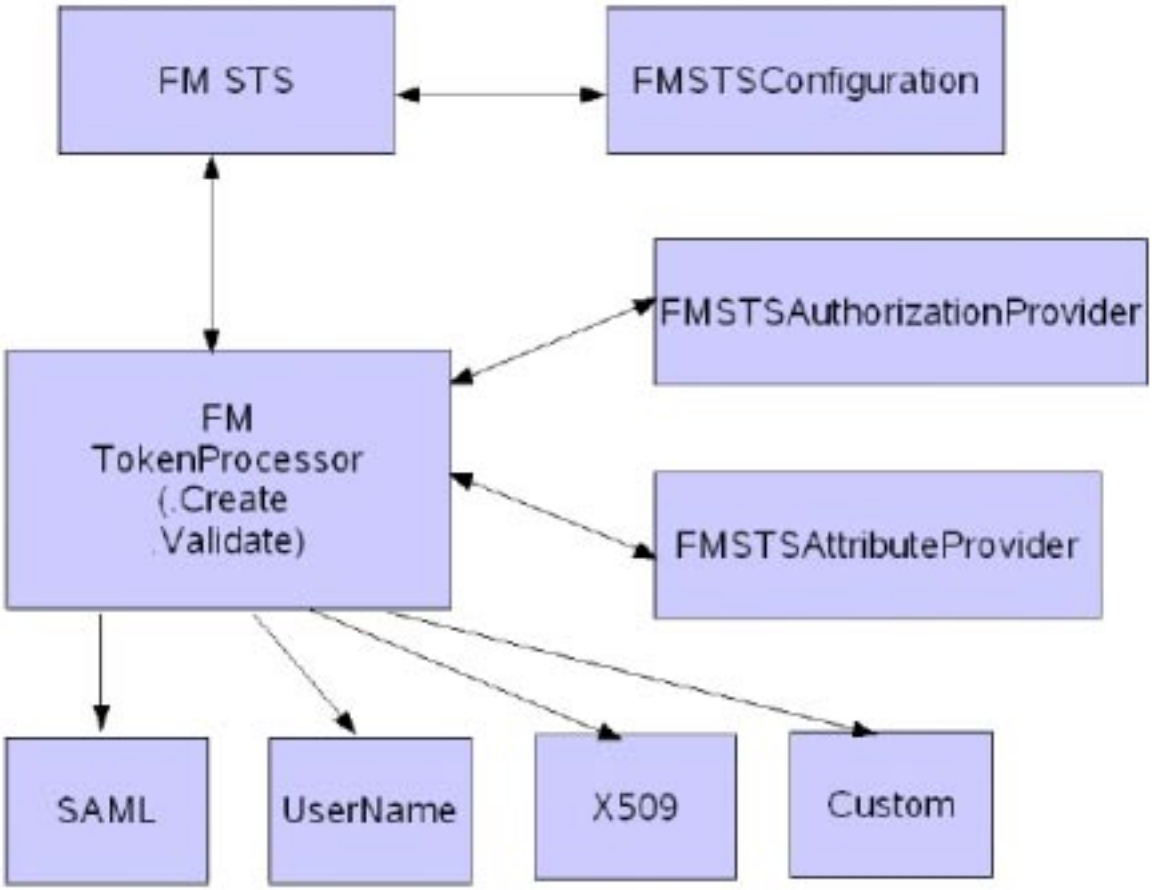


FIGURE 15–5 Security Token Service Design

The Security Token Service supports the following tokens.

- Tokens that can be authenticated with the Security Token Service:
  1. UserName
  2. X509
  3. SAML 1.1
  4. SAML 2.0
  5. Kerberos
- Tokens that can be issued with the Security Token Service:
  1. UserName
  2. X509

3. SAML 1.1
4. SAML 2.0

- End user tokens that can be converted or validated out of the box:
  1. OpenSSO Enterprise `SSOToken` to SAML 1.1 or SAML 2.0 token
  2. SAML 1.1 or SAML 2.0 token to OpenSSO Enterprise `SSOToken`

Additionally, end user tokens can be converted or validated after customization. In this case, the new token is an `On Behalf Of` token (WS-Trust protocol element) carried in the WS-Trust request as part of the SOAP body and not as an authentication token carried as part of the SOAP header. Custom tokens can also be created and sent `On Behalf Of` an end user token for conversion or validation by Security Token Service. To do this, implement the `com.sun.identity.wss.sts.ClientUserToken` interface and put the implemented class name in `AMConfig.properties` on the client side and the global Security Token Service configuration using the OpenSSO console.

Note – You can configure a WSC's agent profile to retrieve tokens from the Security Token Service (using the WS-Trust protocol) or from the Discovery Service (using the Liberty Alliance Project protocol). Based on this configuration, either the Security Token Service client API or the Discovery Service client API (both available through the Client SDK) will take over. For more information, see the *Sun Federated Access Manager 8.0 Administration Guide*.

Security Agents

Remark 15–1 Reviewer **Please review carefully. This was pulled directly from previous doc.**

Security agents (or web service security providers based on the JSR-196 specification) provide message level security, and support Liberty Alliance Project security tokens, Web Services-Interoperability Basic Security Profiles (WS-I BSP) tokens, and proprietary OpenSSO Enterprise session tokens. The agents use an instance of OpenSSO Enterprise for all authentication decisions. Web services requests and responses are passed to the appropriate authentication module using standard Java representations based on the transmission protocol. The following security agents are currently supported.

- "HTTP Security Agent" on page 219
- "SOAP Security Agent" on page 221

Note – In previous Sun documentation, this agent was referred to as an *authentication agent*.

HTTP Security Agent

The HTTP security agent protects the endpoints of a web service that uses HTTP for communication. After the HTTP agent is deployed in a web container on the WSP side, all

HTTP requests for access to web services protected by it are redirected to the login and authentication URLs defined in the OpenSSO Enterprise configuration data on the WSC side. The configurable properties are:

- **com.sun.identity.loginurl=***ossoserver_protocol***://***ossoserver_host***:***ossoserver_port***/ossoserver/UI/L**
- **com.sun.identity.liberty.authnsvc.url=***ossoserver_protocol***://***ossoserver_host***:***ossoserver_port***/oss**

Note – For this release, the HTTP security agents are used primarily for bootstrapping. Future releases will protect web applications.

Figure 15–6 illustrates the interactions described in the procedure below it.

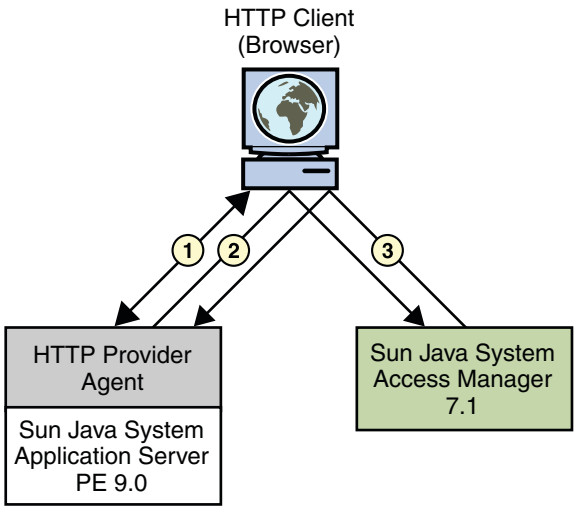


FIGURE 15–6 HTTP Security Agent Process

1. A WSC (user with a browser) makes a request to access a web service protected by an HTTP security agent.
2. The security agent intercepts the request and redirects it (via the browser) to OpenSSO Enterprise for authentication.
3. Upon successful authentication, a response is returned to the web service, carrying a token as part of the Java EE Subject.

   This token is used to bootstrap the appropriate Liberty ID-WSF security profile. If the response is successfully authenticated, the request is granted.

Note – The functionality of the HTTP security agent is similar to that of the Java EE policy agent when used in SSO ONLY mode. This is a non restrictive mode that uses only the OpenSSO Enterprise Authentication Service to authenticate users attempting access. For more information on Java EE policy agents, see the *Sun Java System Access Manager Policy Agent 2.2 User's Guide*.

SOAP Security Agent

The SOAP security agent secures SOAP messages between a WSC and a WSP. It can be configured for use as a security provider on either the WSC server or the WSP server. This initial release encapsulates the Liberty Identity Web Services Framework (Liberty ID-WSF) SOAP Binding Specification (as implemented by OpenSSO Enterprise) and supports the following tokens.

- "Supported Liberty Alliance Project Security Tokens" on page 221
- "Supported Web Services-Interoperability Basic Security Profile Security Tokens" on page 222

Supported Liberty Alliance Project Security Tokens

In a scenario where security is enabled using Liberty Alliance Project tokens, the HTTP client requests (via the WSC) access to a service. The HTTP security agent redirects the request to the OpenSSO Enterprise Authentication Service for authentication and to determine the security mechanism registered by the WSP and obtain the security tokens expected. After a successful authentication, the WSC provides a SOAP body while the SOAP security agent on the WSC side inserts the security header and a token. The message is then signed before the request is sent to the WSP.

When received by the SOAP security agent on the WSP side, the signature and security token in the SOAP request are verified before forwarding the request on to the WSP itself. The WSP then processes it and returns a response, signed by the SOAP security agent on the WSP side, back to the WSC. The SOAP security agent on the WSC side then verifies the signature before forwarding the response on to the WSC. Figure 15–7 illustrates these interactions.

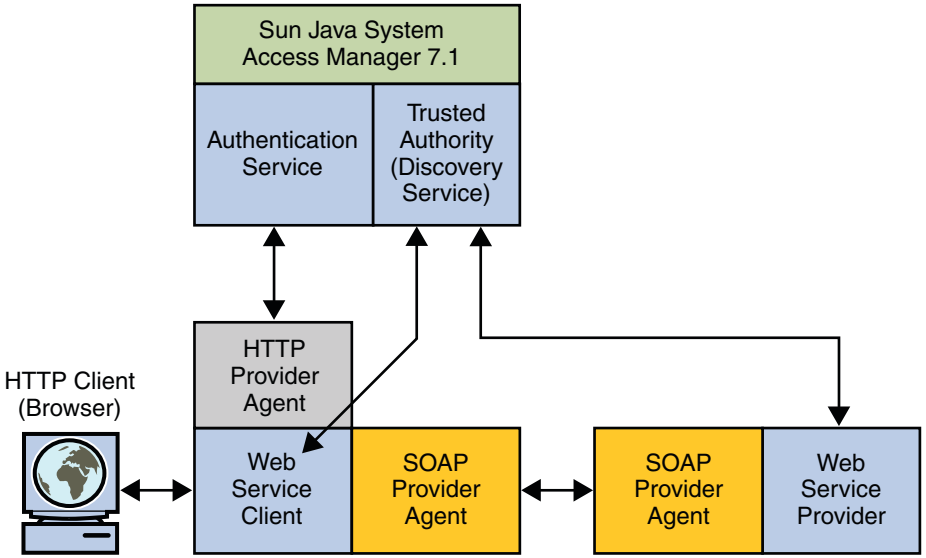


FIGURE 15–7 SOAP Security Agent Process for Liberty Alliance Project Security Tokens

The following Liberty Alliance Project security tokens are supported in this release:

X.509 A secure web service uses a PKI (public key infrastructure) in which the WSC supplies a public key as the means for identifying the requester and accomplishing authentication with the web service provider. Authentication with the web service provider using processing rules defined by the Liberty Alliance Project.

BearerToken A secure web service uses the Security Assertion Markup Language (SAML) *SAML Bearer token* confirmation method. The WSC supplies a SAML assertion with public key information as the means for authenticating the requester to the web service provider. A second signature binds the assertion to the SOAP message This is accomplished using processing rules defined by the Liberty Alliance Project

SAMLToken A secure web service uses the SAML *holder-of-key* confirmation method. The WSC adds a SAML assertion and a digital signature to a SOAP header. A sender certificate or public key is also provided with the signature. This is accomplished using processing rules defined by the Liberty Alliance Project.

Supported Web Services-Interoperability Basic Security Profile Security Tokens

In a scenario where security is enabled using Web Services-Interoperability Basic Security Profile (WS-I BSP) tokens, the HTTP client (browser) requests (via the WSC) access to a service. The SOAP security agent redirects the request to the OpenSSO Enterprise Authentication Service for authentication and to determine the security mechanism registered by the WSP and obtain the expected security tokens. After a successful authentication, the WSC provides a SOAP body while the SOAP security agent on the WSC side inserts the security header and a token. The message is then signed before the request is sent to the WSP.

When received by the SOAP security agent on the WSP side, the signature and security token in the SOAP request are verified before forwarding the request on to the WSP itself. The WSP then processes it and returns a response, signed by the SOAP security agent on the WSP side, back to the WSC. The SOAP security agent on the WSC side then verifies the signature before forwarding the response on to the WSC. Figure 15–8 illustrates the interactions as described.

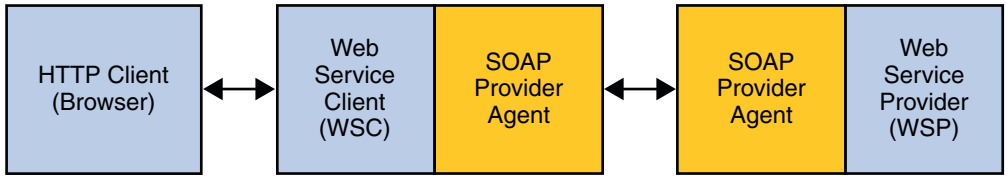


FIGURE 15–8 SOAP Provider Agent Process for WS-I BSP Security Tokens

The following WS-I BSP security tokens are supported in this release.

User Name A secure web service requires a user name, password and, optionally, a signed the request. The web service consumer supplies a username token as the means for identifying the requester and a password, shared secret, or password equivalent to authenticate the identity to the web service provider.

X.509 A secure web service uses a PKI (public key infrastructure) in which the web service consumer supplies a public key as the means for identifying the requester and accomplishing authentication with to the web service provider.

SAML-Holder-Of-Key A secure web service uses the SAML *holder-of-key* confirmation method. The web service consumer supplies a SAML assertion with public key information as the means for authenticating the requester to the web service provider. A second signature binds the assertion to the SOAP payload.

SAML-SenderVouches A secure web service uses the SAML *sender-vouches* confirmation method. The web service consumer adds a SAML assertion and a digital signature to a SOAP header. A sender certificate or public key is also provided with the signature.

Web Services Security Interfaces

Remark 15–2 Reviewer **Please review carefully this information.**

The Web Services Security framework includes the following Java packages:

- `com.sun.identity.wss.provider` provides administrative interfaces for configuration of the WSC and WSP with their respective security mechanisms and Security Token Service configuration. They are called by the security agent during run time, and also by applications that would like to secure messages. On the WSC side, they are called to secure the web service request and to validate any response from the WSP. Similarly, there are interfaces for this functionality on the WSP side. When a WSC is configured to

communicate with the Security Token Service, security mechanisms and security tokens would be obtained from it. When a WSP is configured to communicate with the Security Token Service, its resource offering would be published at the Security Token Service.

Tip – A WSC and a WSP can be associated with one or more Security Token Services.

- `com.sun.identity.wss.security` provides classes that create, manage and represent security tokens and their processing.
- `com.sun.identity.wss.sts` contains a class used to obtain security tokens from the Security Token Service or Liberty Discovery Service.

Note – [Remark 15–3 Reviewer: Which is token conversion interface?] The default implementation of the token conversion interface converts web services tokens to an OpenSSO Enterprise `SSOToken`. This SPI and its implementation will be used by the Security Token Service to validate the web services token against the Policy Service.

The main dependencies and interactions of the Security Token Service and security agents are with the OpenSSO Enterprise Client SDK which includes these packages.

- Security agents bootstrap the Security Token Service or the Liberty Alliance Project Discovery Service using the Client SDK.
- The Client SDK implements XML signing and XML encryption for SOAP requests and responses.
- The Client SDK generates the proprietary `SSOToken` based on security token credentials provided to the WSP. It also sets the `SSOToken` into the container Subject for further authorization processing.
- The Client SDK implements caching for the security tokens generated by the Security Token Service or the Liberty Alliance Project Discovery Service. This improves performance when requesting security tokens.
- The Client SDK implements complete processing (including token insertion, extraction and validation) of SOAP requests and responses.

For more information, see the *Federated Access Manager 8.0 Java API Reference* and the *Sun Federated Access Manager 8.0 Developer's Guide*.

Web Services Security Process

In general, securing web services involves establishing trust between a WSC and a WSP. For identity-based web services specifically (for example, a calendar service), the WSP would have to trust the WSC to authenticate the user, or the WSC would have to include the user's credentials as part of the web service request. distinguishing factor is that identity-based web services authenticate both the WSC and the user's identity. (The user must be authenticated so that the WSC can send the user's *token* to the WSP in a SOAP security header.)

Security agents plug into a web container to provide message level security. They support both Liberty Alliance Project as well as Web Services-Interoperability Basic Security Profiles (WS-I BSP) tokens. Figure 15–9 illustrates this process in detail with the steps provided below it. This process illustrates a scenario when both client and service web containers employ the Java Authentication SPI.

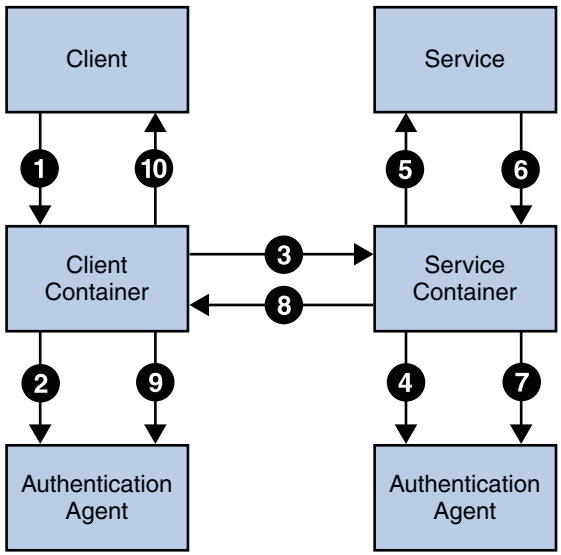


FIGURE 15–9 Web Services Security Process

1. The client browser's attempt to invoke a web service is intercepted by the client's web container.
2. An authentication agent, deployed on the client's web container, is invoked to secure the request (based on the security policy of the web service being invoked).
3. The client's web container sends the secured request message to the web service.
4. An authentication agent, deployed on the web service's web container, is invoked to validate the request and obtain the identity of the caller.

5. Assuming successful authentication, the web service's web container invokes the requested web service.
6. This action (the invocation of the web service) is returned to the web service's web container as a response.
7. The authentication agent, deployed on the web service's web container, is invoked to secure the response message.
8. The web service's web container sends the secured response message to the client.
9. The authentication agent, deployed on the client's web container, is invoked to validate the secured response message.
10. The invocation of the web service is returned to the client browser.

Index

B

C

D

P

R

S

T

U

V

W

X