Not For Publication

# Sun Java System Access Manager 7.1 Technical Overview

Beta

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

Part No: 819–4669–05
December 2006

061115@15490

# Contents

# Preface

Sun Java$^{TM}$ System Access Manager is a component of the Sun Java Enterprise System (Java ES), a set of software components that provide services needed to support enterprise applications distributed across a network or Internet environment. The *Sun Java System Access Manager 7.1 Technical Overview* describes Access Manager features, explains what Access Manager does, and explains how Access Manager 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 Java System servers and software. Readers of this guide should be familiar with the following concepts and technologies:

- Deployment platform: Solaris$^{TM}$ or Linux operating system
- Web container that will run Access Manager: Sun Java System Application Server, Sun Java System Web Server, BEA WebLogic, or IBM WebSphere Application Server
- Technical concepts: Lightweight Directory Access Protocol (LDAP), Java technology, JavaServer Pages$^{TM}$ (JSP) technology, HyperText Transfer Protocol (HTTP), HyperText Markup Language (HTML), and eXtensible Markup Language (XML)

## Related Books

Related documentation is available as follows:

- “Access Manager Core Documentation” on page 8
- “Sun Java Enterprise System Product Documentation” on page 9

### Access Manager Installation Instructions

For detailed information about installing Access Manager, see the following Java Enterprise System (JES) documents:

- *Sun Java Enterprise System 2006Q4 Release Notes*
- *Sun Java Enterprise System 2006Q3 Deployment Planning Guide*

- *Sun Java Enterprise System 5 Installation Guide for UNIX*
- *Sun Java Enterprise System 2006Q3 Upgrade Guide*

## Access Manager Core Documentation

The Access Manager core documentation set contains the following titles:

- The *Sun Java System Access Manager 7.1 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 Java System Access Manager 7 Technical Overview* (this guide) provides an overview of how Access Manager components work together to consolidate access control functions, and to protect enterprise assets and web-based applications. It also explains basic Access Manager concepts and terminology.
- The *Sun Java System Access Manager 7.1 Deployment Planning Guide* provides planning and deployment solutions for Sun Java System Access Manager based on the solution life cycle
- The *Sun Java System Access Manager 7.1 Postinstallation Guide*provides information for configuring Access Manager after running the Java Enterprise System (JES) installer.
- The *Sun Java System Access Manager 7.1 Performance Tuning Guide* provides information on how to tune Access Manager and its related components for optimal performance.
- The *Sun Java System Access Manager 7.1 Administration Guide* describes how to use the Access Manager console as well as manage user and service data via the command line interface.
- The *Sun Java System Access Manager 7.1 Federation and SAML Administration Guide* provides information about the Federation module based on the Liberty Alliance Project specifications. It includes information on the integrated services based on these specifications, instructions for enabling a Liberty-based environment, and summaries of the application programming interface (API) for extending the framework.
- The *Sun Java System Access Manager 7.1 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 Java System Access Manager 7.1 C API Reference* provides summaries of data types, structures, and functions that make up the public Access Manager C APIs.
- The *Sun Java System Access Manager 7.1 2006Q4 Java API Reference* provides information about the implementation of Java packages in Access Manager.
- The *Sun Java System Access Manager Policy Agent 2.2 User's Guide* provides an overview of the policy functionality and the policy agents available for Access Manager.

Updates to the *Release Notes* and links to modifications of the core documentation can be found on the Access Manager page at the Sun Java Enterprise System documentation web site. Updated documents will be marked with a revision date.

## Sun Java Enterprise System Product Documentation

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

- Directory Server
- Web Server
- Application Server
- Web Proxy Server

## Related 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.

## 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/`)

## Typographic Conventions

The following table describes the typographic conventions 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.<br>Use `ls -a` to list all files.<br>`machine_name% you have mail.` |

**TABLE P–1** Typographic Conventions *(Continued)*

| Typeface | Meaning | Example |
|---|---|---|
| **`AaBbCc123`** | What you type, contrasted with onscreen computer output | `machine_name%` **`su`**<br>`Password:` |
| *aabbcc123* | Placeholder: replace 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 | Read Chapter 6 in the *User's Guide*.<br>A *cache* is a copy that is stored locally.<br>Do *not* save the file.<br>**Note:** Some emphasized items appear bold online. |

# Shell Prompts in Command Examples

The following table shows the default UNIX® system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

**TABLE P–2** Shell Prompts

| Shell | Prompt |
|---|---|
| C shell | `machine_name%` |
| C shell for superuser | `machine_name#` |
| Bourne shell and Korn shell | `$` |
| Bourne shell and Korn shell for superuser | `#` |

# 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 document title and part number. The part number is a seven-digit or nine-digit number that can be found on the title page of the book or at the top of the document.

For example, the title of this book is *Sun Java System Access Manager 7.1 Technical Overview*, and the part number is 819-2135–10.

CHAPTER 1

# Introduction to Access Manager

Sun Java$^{TM}$ System Access Manager 7.1 2006Q3 integrates authentication and authorization services, policy agents, and identity federation to provide a comprehensive solution for protecting your network resources. Access Manager prevents unauthorized access to web service applications and web content. This chapter provides an overview Access Manager features and architecture.

Topics in this chapter include:

## An Access Management Paradigm

Think of all the different types of information a company must store and be able to make available through its enterprise. Now consider the various enterprise users who must make use of that information in order for the company's business to run smoothly. For example, the following are routine information transactions that occur every day in a typical company:

- An employee looks up a colleague's phone number in the corporate phone directory.
- A manager looks up the salary histories of her reports to help determine an individual's merit raise.
- An administrative assistant adds a new hire to the corporate database, which triggers 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 the company's website and looks for a product in the company's online catalog.
- A vendor submits an online invoice to the company's accounting department.

In each of these examples, the company must determine who is allowed to view its information or use its applications. Some information such as the company's product descriptions and advertising can be made available to everyone, even the public at large, in the company's online catalog. Other information such as accounting and human resources information must be restricted to only employee use. And some internal information is appropriate to share with partners and suppliers, but not with customers.

## The Problem

Many enterprises grant access to information on a per-application basis. For example, an employee might have to set up a user name and password to access the company's health benefits administration website. The same employee must use a different user name and password to access the Accounting Department online forms. Within the same enterprise, a customer sets up a user name and password to access the public branch of the company website. For each website or service, an administrator must convert the enterprise user's input into a data format that the service can recognize. Each service added to the enterprise must be provisioned and maintained separately.

## The Solution

Access Manager reduces the administrative costs and eliminates the redundant user information associated with per-application solutions. Access Manager enables an administrator to assign specific rules or policies governing which information or services each user can access. Policy agents are deployed on application or web servers to process HTTP requests and to enforce active policies.

Together, a user's information and associated access policies comprise the user's enterprise identity. Access Manager makes it possible for a user to access many resources in the enterprise with just one identity.

# What Access Manager Does

When an enterprise user or an external application tries to access content stored on a company's web server, the Access Manager policy agent intercepts the request and directs it to the Access Manager server. Access Manager asks the user to present credentials such as a username and password. If the credentials match those stored in the appropriate identity repository, Access Manager determines that the user's credentials are authentic.

Following authentication, the Access Manager policy agent evaluates the policies associated with the user's identity for authorization. Policies are created using Access Manager and identify which users (or groups of users) are authorized to access a resource, specifying conditions under which this authorization is valid. Finally, based upon policy evaluation results, the policy agent either grants or denies the user access to the information. The figure Figure 1–1 below illustrates one way Access Manager can be configured to act as the gatekeeper to a company's information resources.

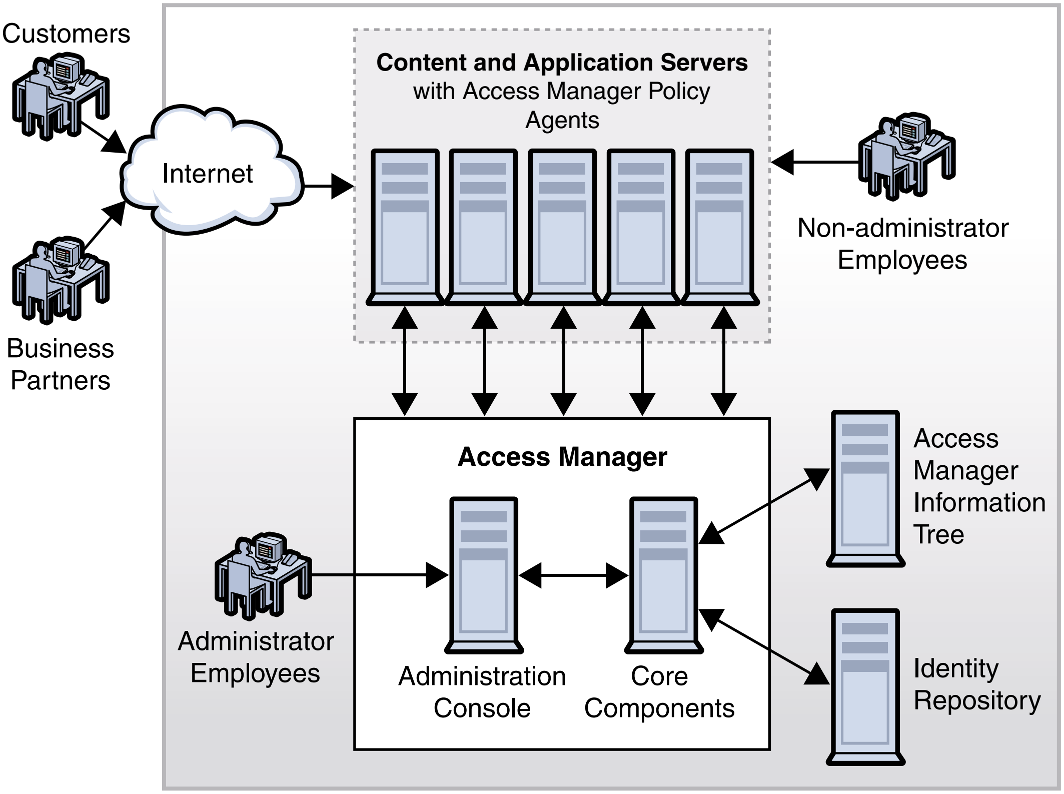


FIGURE 1–1 Access Manager as the Gateway to a Company's Enterprise Resources

Access Manager integrates the following features into a single product that can be viewed in a single administration console:

- "Authentication Service" on page 13
- "Policy Service" on page 14
- "User Session Management" on page 14
- "SAML Service" on page 14
- "Identity Federation Service" on page 15
- "Logging" on page 15

## Authentication Service

Authentication is the first step in determining whether a user is allowed to access a resource protected by Access Manager. The Access Manager Authentication service verifies that a user really is the person he claims to be. Authentication service consists of the following components: plug-in modules, a framework for connecting plug-in modules, a core authentication component, a web service interface, and client APIs. Authentication Service interacts with the Authentication database to validate user credentials, and interacts with Identity Repository Management plug-ins to retrieve user profile attributes. When 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*.

## Policy Service

Authorization is the process by which Access Manager evaluates policies associated with a user's identity, and determines whether an authenticated user has permission to access a protected resource. Access Manager Policy service enables authorization to take place. Policy service consists of the following components: policy plug-ins, a framework for connecting policy plug-ins, a core policy component, a web service interface, and client APIs. Policy service interacts with Access Manager service configurations, delegation service, and identity repository plug-ins to verify that the user has access privileges from a recognized authority.

## User Session Management

An Access Manager user session is the interval between the moment a user logs in to a network resource protected by Access Manager, and the moment the user logs out of the resource. During the user session, Access Manager session service maintains information about the user's interaction with various applications the user accesses. Access Manager uses this information to enforce time-dependent rules such as timeout limits. Also during the user session, Access Manager provides continuous proof of the user's identity. This continued proof of identity enables the user to access multiple enterprise resources without having to provide credentials each time.

The Access Manager Session Service 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.
- **Single sign-on (SSO) session.** The user provides credentials once, and can then access multiple applications within the same DNS domain.
- **Cross-domain SSO session.** The user provides credentials once, and can then access applications among multiple DNS domains.

## SAML Service

Access Manager uses the Security Assertion Markup Language (SAML), an XML based framework for exchanging security information. While Access Manager User Session service enables single sign-on sessions among different DNS domains within the same intranet, SAML service enables cross-domain sign-on (CDSSO) sessions among different business domains. Using the SAML protocol, business partners can securely exchange authentication and authorization information over the Internet. Access Manager SAML service consists of a web service interface, a SAML core component, and a SAML framework that web services can connect to.

## Identity Federation Service

Identity federation allows a user to consolidate the many local identities he has configured among multiple service providers. With one federated identity, the user can log in at one service provider's site and move to an affiliated service provider site without having to re-authenticate or re-establish his identity. Identity Federation service works with SAML service to enable single sign-on sessions among business partners over the Internet. Identity Federation services consists of a web service interface, a core Identity Federation component, and an Identity Federation Framework that complies with the Liberty Alliance Project specifications.

## Logging

When a user logs in to a resource protected by Access Manager, the Logging component logs information about the user's activity. You can write custom log operations and customize log plug-ins to generate log reports for auditing purposes.

# How Access Manager Works

When Access Manager starts up, it initializes the Access Manager information tree with configuration data. The configuration data comes from Access Manager service plug-ins including Authorization, Policy, Identity Repository Management, and Service Configuration plug-ins. By default, the Access Manager information tree resides in Sun Java System Directory Server, the same data store as the default identity repository.

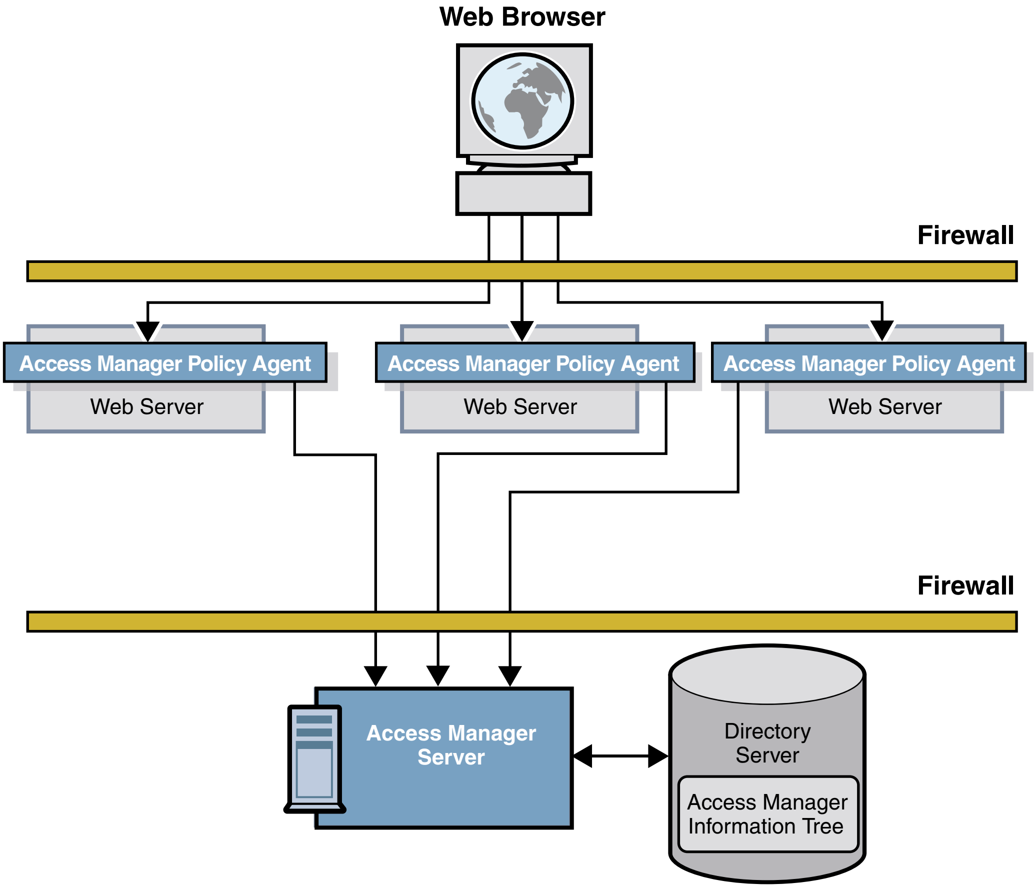

Web Browser
Firewall
Access Manager Policy Agent
Web Server
Access Manager Policy Agent
Web Server
Access Manager Policy Agent
Web Server
Firewall
Access Manager Server
Directory Server
Access Manager Information Tree

**FIGURE 1–2** Basic Access Manager Installation

When a browser sends a request to access content or an application on a protected resource, Access Manager immediately binds to the appropriate Identity Repository to obtain user information. The user information may include definitions for roles, realms, user ids, and so forth. At the same time, a Policy Agent installed on the protected resource intercepts the initial HTTP request and examines the request. If no session token is found, the Policy Agent contacts the Access Manager server. Then Access Manager invokes authentication and authorization processes.

# Access Manager Architecture

Access Manager uses a Java technology-based architecture for scalability, performance, and ease of development.

Access Manager leverages industry standards including HyperText Transfer Protocol (HTTP), eXtensible Markup Language (XML), Simple Object Access Protocol (SOAP), and Security Assertions markup Language (SAML) specification. The Access Manager internal architecture is multi-layered and includes a presentation layer, web services, core components, an integrated framework, and a plug-ins layer.

Custom applications access the Access Manager web services through the Access Manager client application programming interfaces (APIs) installed on each protected resource. Custom plug-in modules interact with the Access Manager service provider interfaces (SPIs) and plug-ins framework. The plug-in modules retrieve information from the Access Manager information tree and deliver required information to other plug-ins when necessary, and to the Access Manager framework for data processing.

## Web Services

Web services follow a standardized way of integrating Web-based applications using XML, SOAP, and other open standards over an Internet protocol backbone. Web services enable applications from various sources to communicate with each other because web services are not tied to any one operating system or programming language. Businesses use web services to communicate with each other and with clients without having to know detailed aspects of each other's IT systems.

Access Manager provides web services that use XML and SOAP over HTTP. Access Manager web services are designed to be centrally provided in your network for convenient access by your client applications. The following table summarizes the web services provided in Access Manager.

**TABLE 1–1** Access Manager Web Services

| Web Service Name | Description |
|---|---|
| Authentication | Verifies that a user really is the person he claims to be. |
| Policy (Authorization) | Evaluates policies associated with a user's identity, and determines whether an authenticated user has permission to access a protected resource. |
| SAML | Enables single sign-on sessions among different business domains. Allows business partners to securely exchange authentication and authorization information over the Internet. |
| Identity Federation | Enables a user to log in at one service provider's site and move to an affiliated service provider site without having to re-authenticate or re-establish his identity. |

**TABLE 1–1** Access Manager Web Services *(Continued)*

| Web Service Name | Description |
|---|---|
| Session | Maintains information about the user's interaction with various applications the user accesses. |

Access Manager uses both eXtensible Markup Language (XML) files and Java interfaces to and manage web services and web service configuration data. An Access Manager XML file is based on a structure defined in a corresponding DTD file. The DTD file defines the elements and qualifying attributes needed to form a valid XML document. Access Manager includes DTD files that define the structure for the different types of XML files it uses. The DTDs are located in AccessManager-base/`SUNWam/dtd`. The file `sms.dtd` defines the structure for all XML service files. All XML service files are located in `/etc/opt/SUNWam/config/xml`.

⚠ **Caution –** Do not modify any of the Access Manager DTD files. The Access Manager APIs and their internal parsing functions are based on the default definitions. Alterations to the DTD files may hinder the operation of Access Manager.

## Core Components and Services that Power Access Manager

The core components provide the logic that performs the main Access Manager functions. The core components work with services that run within Access Manager. These internal services process data solely for use by Access Manager. The following table lists the core Access Manager components and internal services along with brief descriptions of what they do.

**TABLE 1–2** Access Manager Core Components and Internal Services

| Core Component or Service | What it Does |
|---|---|
| Authentication component | Validates user's credentials and verifies that the user is who he claims to be. |
| Authorization (Policy) component | Evaluates policies to determine whether the user has permission to access the requested resource. |
| Security Assertion Markup Language (SAML) component | Provides a protocol-based alternative to using cookies for performing a single sign-on session. |
| Identity Federation component | Enables user to access resources provided by multiple business partners in a single sign-on session. |
| User Session Management component | Maintains information about user session, and enforces timeout limits. Provides continued proof of identity to enable single sign-on sessions. |

TABLE 1–2 Access Manager Core Components and Internal Services *(Continued)*

| Core Component or Service | What it Does |
|---|---|
| Logging service | Tracks a user's interactions with web applications. Creates log messages to form an audit trail of important events within the system. |
| Naming service | Enables a client to locate other Access Manager services such as User Session Management Service, Logging Service, Policy Service, and so forth. Defines URLs used to access these internal services. |
| Platform service | Manages configurable attributes used in an Access Manager deployment. |
| Client Detection service | Detects the client type of the browser being used to access the Access Manager application. Client types include HTML, WML, and other protocols. |

## Client APIs

Enterprise resources cannot be protected by Access Manager until you install Access Manager client APIs on the Web Server or Application Server that you want to protect. (The client APIs are automatically installed when you install a policy agent.) The client APIs allow you to customize an application by enabling communication with Access Manager for retrieving user, session, and policy data.

## Access Manager Framework

The framework layer is where the Access Manager business logic is implemented. Each core component uses its own framework to retrieve customer data from the plug-in layer and to provide data to the core components. The Access Manager framework integrates all of these frameworks to form one layer in the architecture that is accessible to all core components and all Access Manager plug-ins.

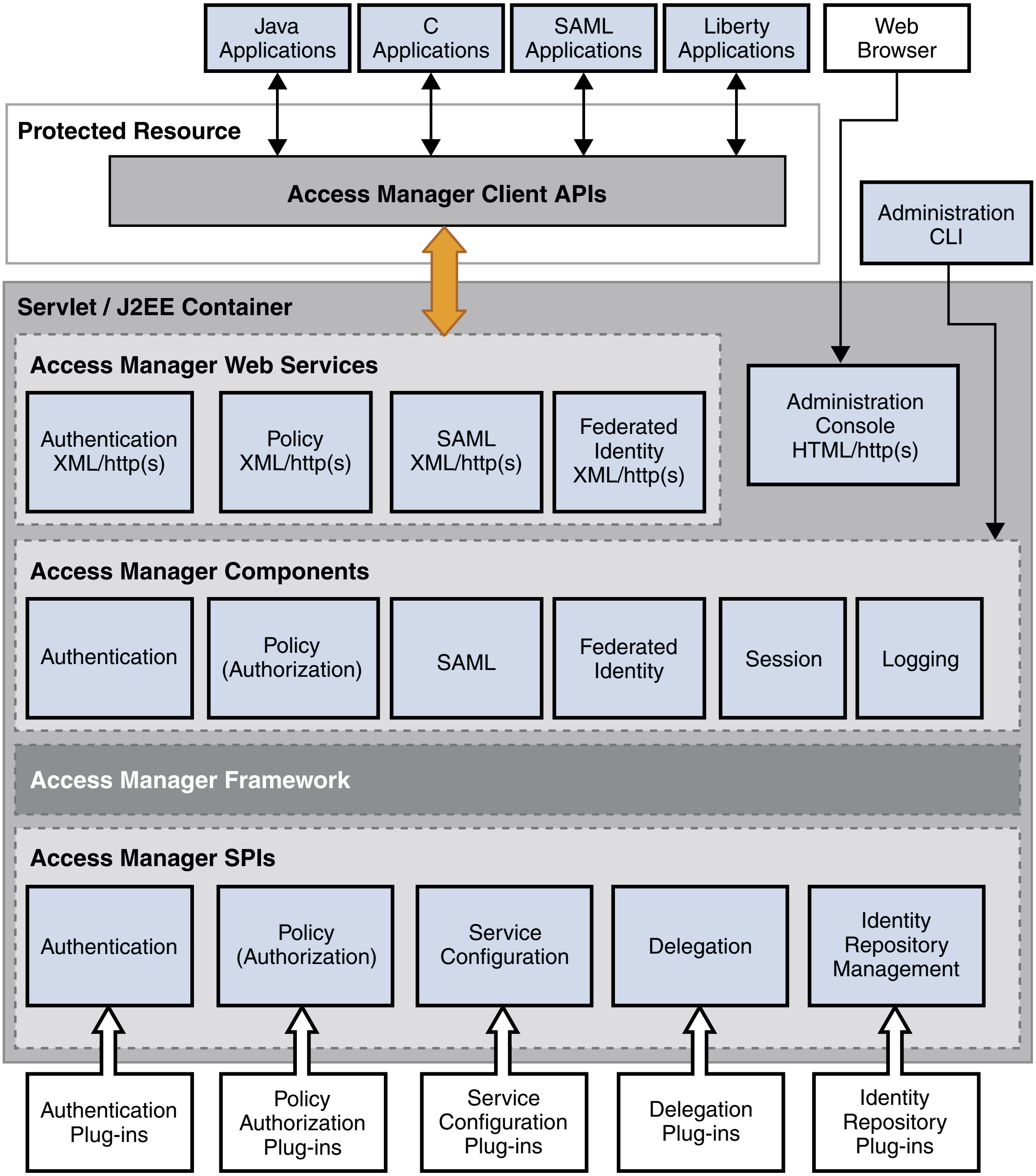


FIGURE 1–3 Access Manager Internal Architecture

## Plug-ins Layer

The Access Manager SPIs work with plug-ins to provide customer data to the framework for back-end processing. Some customer data comes from external data base applications such as identity repositories. Some customer data come from Access Manager plug-ins. You can develop custom plug-ins to work with Access Manager SPIs.

For a complete listing of Access Manager SPIs, see the Javadoc. The following table lists the plug-ins that are installed with Access Manager and a brief description what each plug-in does.

**TABLE 1–3** Access Manager Plug-ins

| Plug-in | Description |
|---|---|
| Authentication | Accesses user data in a specified identity repository to determine if user's credentials are valid. |
| Policy | Aggregates policies and rules to determine whether a user is authorized to access a protected resource. |
| Service Configuration | Manages configuration data used in each core component framework: authentication, authorization, SAML, session, logging, and identity federation. Provides configuration data to any Access Manager plug-in or component that needs the data. |
| Delegation | Aggregates policies and rules to determine the scope of a network administrator's authority. |
| Identity Repository Management | Authenticates identities and returns identity information such as user attributes and membership status. |
| AM SDK | Creates and modifies users and stores information in the user branch of the identity repository. Implements user management APIs used in previous Access Manager releases. |

## Access Manager Policy Agents

You install an Access Manager Policy Agent on a protected resource to enforce the policy decisions determined by the Policy Service. The policy agent intercepts requests from applications, and redirects the requests to Access Manager for authentication. Once the user is authenticated, the policy agent communicates with the Policy Service. The policy agent allows the user access or denies the user access depending upon the result of policy evaluation.

## Access Control Realms

In Access Manager an access control realm is a group of authentication properties and authorization policies you can associate with a user or group of users. Realm data is stored in a proprietary information tree that Access Manager creates within a data store you specify. The Access Manager framework aggregates policies and properties contained in each realm within the Access Manager information tree.

By default, Access Manager automatically inserts the Access Manager information tree as a special branch in Sun Java Enterprise System Directory Server, apart from the user data.

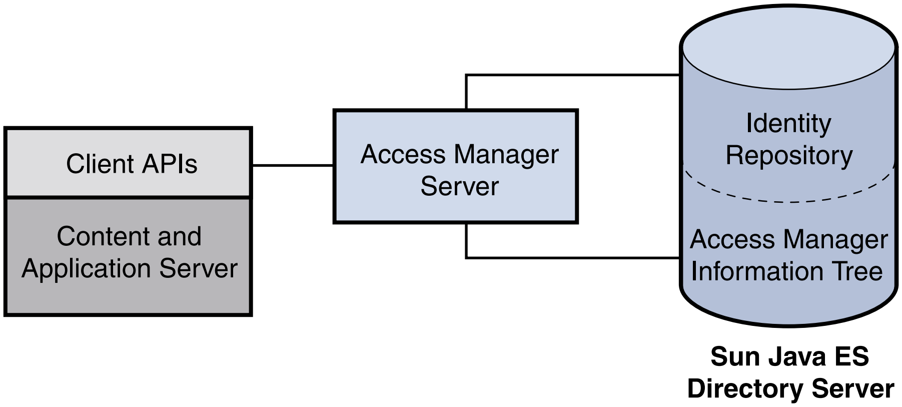

Client APIs
Content and Application Server
Access Manager Server
Identity Repository
Access Manager Information Tree
Sun Java ES Directory Server

**FIGURE 1–4** Default Configuration for Access Manager Information Tree

You can use access control realms while using any user database. The following figure illustrates the Access Manager information tree configured in a separate data store from the identity repository.

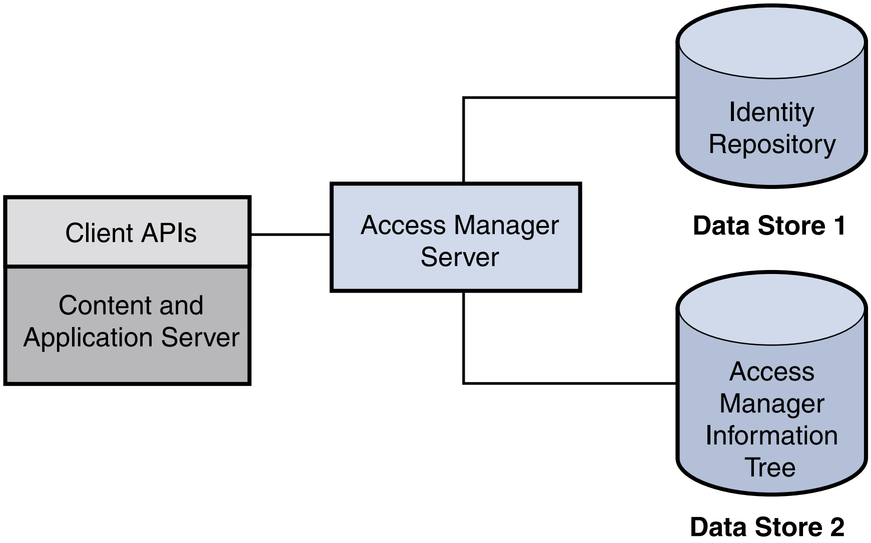


**FIGURE 1–5** Access Manager Information Tree Configured in Second Data Store

When a user logs into an application, Access Manager plug-ins retrieve all user information and access information that Access Manager needs to form a temporary, virtual user identity. Authentication service and Policy service use the virtual user identity to authenticate the user and to enforce authorization policies. The virtual user identity is destroyed when the user's session ends.

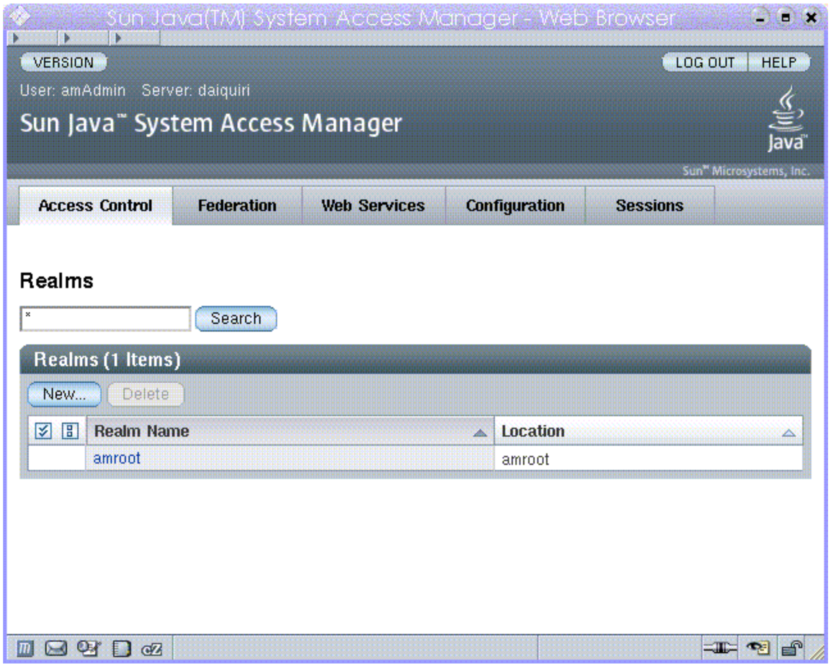
## Realm Mode and Legacy Mode

When you install Access Manager, you are asked to choose either Realm Mode or Legacy Mode.

Realm mode is based on the Access Manager information tree and Identity Repository Management Service described in the previous sections. Realm mode is appropriate in most new Access Manager deployments where you want to keep identity repositories independent of access management, or where you cannot maintain user data within the required object classes of Sun Java System Directory Server.

If you choose Realm Mode at installation, then after installation your identity repositories can exist in any of the following configurations:

- In the same Directory Server instance and the same suffix as the Access Manager information tree.
- In the same Directory Server instance but in a different suffix as the Access Manager information tree.
- In a different directory server instance from the Access Manager information tree.



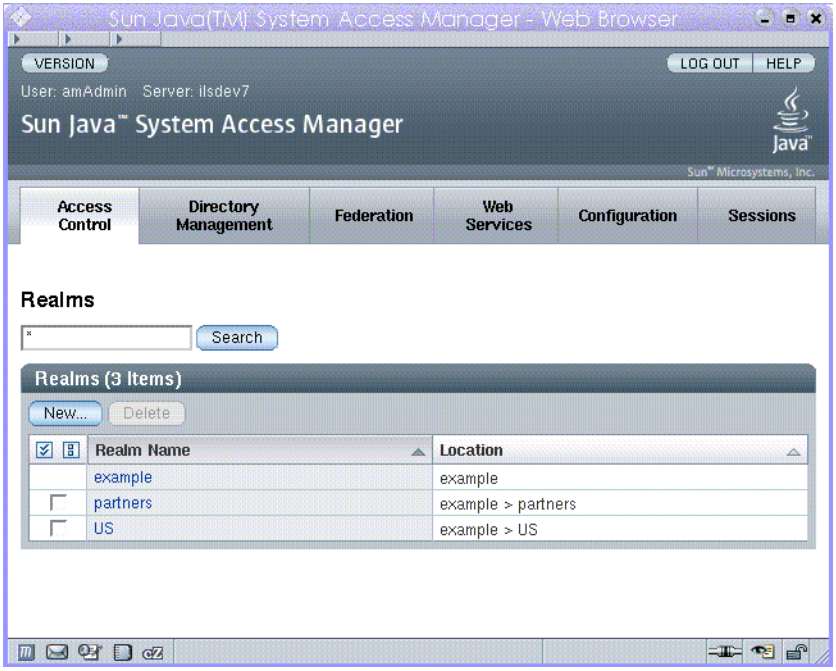
| Realm Name | Location |
|---|---|
| amroot | amroot |

FIGURE 1–6 Realm Mode User Interface

Legacy Mode is based on the Access Manager 6.3 architecture. This legacy Access Manager architecture uses the LDAP directory information tree (DIT) that comes with Sun Java System Directory Server. In Legacy Mode, both user information and access control information are stored in LDAP organizations. When you choose Legacy Mode, an LDAP organization is the equivalent of an access control realm. Realm information is integrated within LDAP organizations.

Legacy Mode is appropriate in deployments where you want to use Access Manager user management. Legacy Mode is typically used in deployments where Access Manager is built upon Sun Java System Portal Server or other Sun Java System communication products that require the use of Sun Java System Directory Server as the central identity repository.

If you choose Legacy Mode during installation, then after installation the top-level ream resides in the same Directory Server branch as the Access Manager information tree, and user information is intermingled with access information.



Realms

Realms (3 Items)

| ☑ | Realm Name | Location |
|---|---|---|
| | example | example |
| □ | partners | example > partners |
| □ | US | example > US |

FIGURE 1–7 Legacy Mode User Interface

The following table compares realm mode and legacy mode.

TABLE 1–4 Comparison of Realm and Legacy Modes

| | Realm Mode | Legacy Mode |
|---|---|---|
| Supports all new Access Manager 7.1 features. | Yes | Yes |
| Supports identity repositories in Sun Java System Directory Server and in other data stores. | Yes | Yes |
| Supports Access Manager 6 user management features. | No | Yes |
| Can coexist with Access Manager 6 2005Q1 in multiple-server installations. | No | Yes |
| Before installation, identity repository can exist in Sun Java Directory Server . | Yes | Yes |
| Before installation, identity repository can exist in an LDAP version 3 compliant directory server. | Yes | No |

For more information about realm and legacy modes, see the Sun Java System Access Manager 7 Release Notes.

# Identity Repository Framework

An identity repository is a database where you can store user attributes and user configuration data. Previous versions of Access Manager relied on Sun Java System Directory Server as the only supported identity repository and the only supported software for creating, managing, and storing user data.

Access Manager provides an identity repository plug-in that connects to an identity repository framework. This model enables you to view and retrieve Access Manager user information without having to make changes in your existing user database. The Access Manager framework integrates data from the identity repository plug-in with data from other Access Manager plug-ins to form a virtual identity for each user. Access Manager can then use the universal identity in authentication and authorization processes among more than one identity repository. The virtual user identity is destroyed when the user's session ends.

You can configure the Identity Repository Management Service per realm to use its own list of Identity Repositories.

Using realm-based configuration, you can specify a single Identity Repository that will store service configurations for both users and roles. The Identity Repository Service provides a list of Identity Repositories that can provide user attributes to Policy, SAML , and Liberty services. The Identity Repository Services pluggable interface combines attributes obtained from different repositories. Identity Repository plug-ins provide interfaces to create, read , edit, and delete objects such as Realm, Role, Group, User, and Agent.

The default identity repository plug-in is designed to work with Sun Java Directory Server which is based on LDAP. In previous Access Manager versions, the functionality of this default plug-in was provided by the AMSDK component. In Access Manager 7.1, the AMSDK functionality still exists, but now in plug-in form.

## Delegation Plug-In

The Delegation plug-in works together with the Identity Repository plug-in to determine a network administrator's scope of privileges. Default administrator roles are defined in the Identity Repository plug-in. The Delegation plug-in forms rules that describe the scope of privileges for each network administrator, and also specifies the roles to which the rules apply. The following is a list of roles defined in the Identity Repository, and the default rule the Delegation plug-in applies to each role.

**TABLE 1–5** Access Manager Roles and Scope of Privileges

| Identity Repository Role | Delegation Rule |
|---|---|
| Realm Administator | Can access all data in all realms of the Access Control information tree. |
| Subrealm Administrator | Can access all data within a specific realm of the Access Control information tree. |
| Policy Administrator | Can access all policies in all realms of the Access Control information tree. |
| Policy Realm Administrator | Can access policies only within the specific realm of the Access Control information tree. |

Authentication service and Policy service use the aggregated data to perform authentication and authorization processes. The Delegation plug-in code is not public in Access Manager.

## Service Configuration Plug-Ins

The Service Configuration plug-in stores and manages data required by other Access Manager plug-ins. In previous versions of Access Manager, the functionality provided by the Service Configuration plug-in was known as the Service Management Service (SMS).

# CHAPTER 2

# User Session Management and Single Sign-On

This chapter explains how the Access Manager Session Service works with other core Access Manager components to process HTTP requests and to manage user session data. The chapter traces events in a basic user session, a single sign-on session (SSO), and a cross-domain single sign-on session (CDSSO) to give you an overview of Access Manager's features and process flows.

This chapter contains the following sections:

## Overview of Access Manager User Sessions

The Session Service in Sun Java System Access Manager tracks a user's interaction with web applications. For example, the session service maintains information about how long a user has been logged in to Access Manager, and enforces time-out limits when necessary.

Session Service performs the following actions:

- 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 (SSO) and cross-domain single sign-on (CDSSO) among applications external to Access Manager.

A *user session* is the interval between the moment a user logs in to Access Manager, and the moment the user logs out of Access Manager. In a typical user session, an employee attempts to access the corporate benefits administration application. The application is protected by Access Manager, and Access Manager prompts the user for a username and password. First, Access Manager *authenticates*, or verifies that the user is who he says he is. Access Manager then allows the user access to the application.

In the same user session (without logging out of the health benefits application), the same employee attempts to access the corporate expense reporting application. The expense reporting application is also protected by Access Manager. In this second transaction, the Access Manager session service provides continued proof of the user's authentication, and the employee is automatically allowed to access the expense reporting application. The employee has accessed more than one service in a single user session without having to re-authenticate. This functionality is called *Single Sign-On* (SSO). When SSO occurs among applications in more than one DNS domain, the functionality is called *Cross-Domain Single Sign-On* (CDSSO).

# Cookies and Session Objects

The Session Service uses cookies and creates session objects to store information about a user session. In an Access Manager user session, session service is most commonly used to enforce timeout limits. For example, you can use session service to configure the Access Manager application so that the user is automatically logged out after x minutes of Access Manager inactivity. The session service can also be used to store additional information to be used by other applications.

## Cookies are Identifiers

A *cookie* is an information packet generated by a web server and passed to a web browser. In Access Manager, a cookie serves as user identifier. The fact that a web server generates a cookie for a user does not guarantee that the user is allowed access to protected resources. The cookie simply points to user information in a data store.

Cookies are domain-specific. For example, a cookie generated by a web server within DomainA cannot be used by a web server in DomainB. Cookies can be passed only between servers in the same domain in which the cookie was set. Similarly, servers can set cookies only on servers within in their own domain.

## Objects in the Session Data Structure

When a user logs in and is successfully authenticated, or verified to be who the user says he is, the user is assigned a session. A *session* is a data structure that contains maximum timeout limits and information about caching time limits. Session service also generates a session token for the new session data structure. The *session token*, also known as a *sessionID*, is an encrypted, unique string that identifies the specific session instance. If the sessionID is known to a protected resource such as an application, the application can access the session and all user information contained in it.

Minimally, an Access Manager session data structure stores the following information about a user session:

| | |
|---|---|
| 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. |

Internally, these session attributes are used to enforce Access Manager timeout limits.

A session can also contain additional attributes and 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 Access Manager to include the following types of information in a session:

- Fixed session attributes
- Protected properties
- Custom properties

For a detailed summary of information that can be included in a session, see the *Sun Java System Access Manager 7.1 Developer's Guide*.

# Policy Agents

Policy agents are programs that police the web server or application server that hosts protected resources. When a user requests access to a protected resource such as a server or an application, the policy agent intercepts the request and redirects the request to Access Manager authentication service. The policy agent also enforces the user's assigned policies. Policy agents are an integral part of SSO and CDSSO sessions.

Access Manager supports two types of policy agents:

- Web agent– Enforces URL-based policy for C applications.

- J2EE/Java agent– Enforces URL-based policy and J2EE-based policy for Java applications on J2EE containers.

Both types of agents are available for you to install as programs separate from Access Manager. For comprehensive information about policy agents and how to install them, see the *Web Policy Agent GuideSun Java System Access Manager Policy Agent 2.2 Guide for Sun Java System Web Server 6.1* and *J2EE Policy Agent GuideSun Java System Access Manager Policy Agent 2.2 Guide for Sun Java System Application Server 8.1*.

# Basic User Session

When Access Manager policy agents are implemented, by default all HTTP requests are implicitly denied unless explicitly allowed by the presence of two things: 1) a valid session, and 2) policy allowing access. You can modify the default configuration so that Access Manger implicitly allows access unless explicitly denied. For detailed information on configuring Session Service, see "The Current Sessions Interface" in *Sun Java System Access Manager 7.1 Administration Guide*.

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 Access Manager. In these examples, the server which hosts an application is protected by an Access Manager policy agent. The Basic User Session includes the following phases:

- "Initial HTTP Request" on page 32
- "User Authentication" on page 34
- "Session Validation" on page 36
- "Policy Evaluation and Enforcement" on page 38
- "Results Logging" on page 40
- "Session Termination" on page 45

## Initial HTTP Request

When a user initiates a user session by using a browser to log in to a web-based application, the events in the following illustration occur. The accompanying text describes the process.

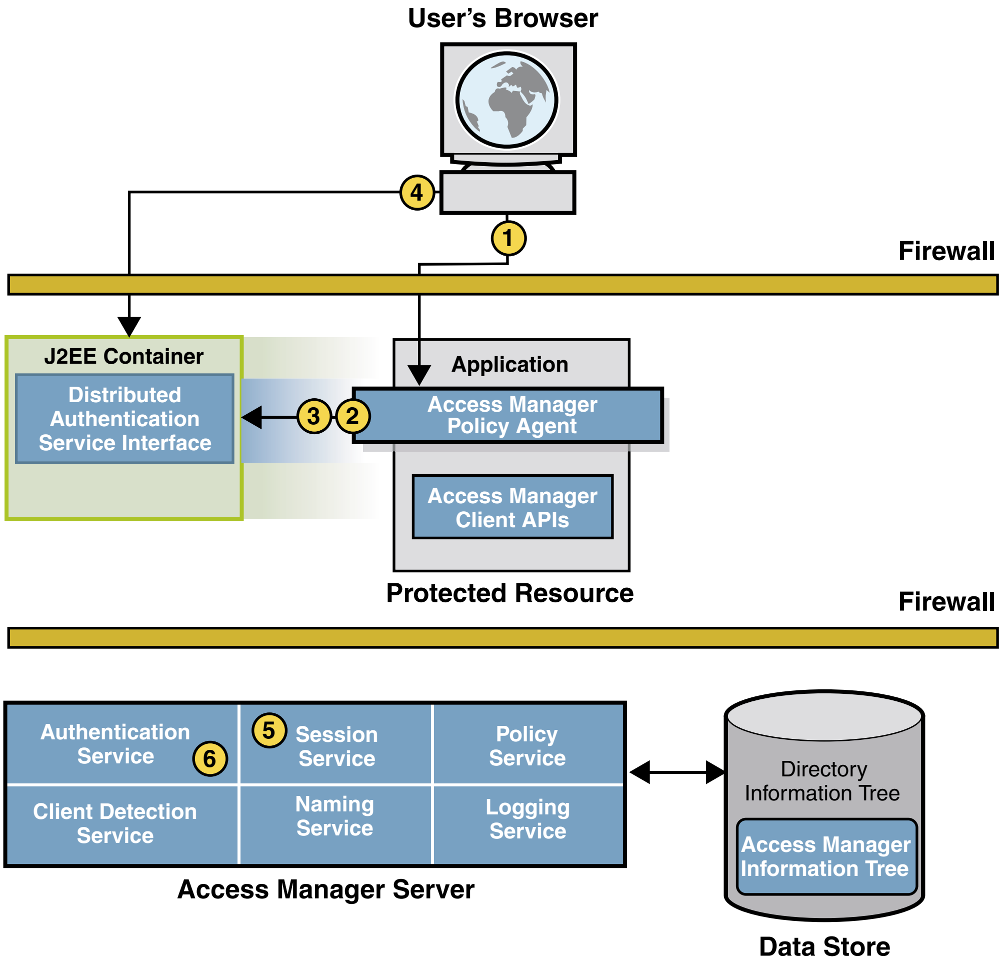


FIGURE 2–1 Initial HTTP Request

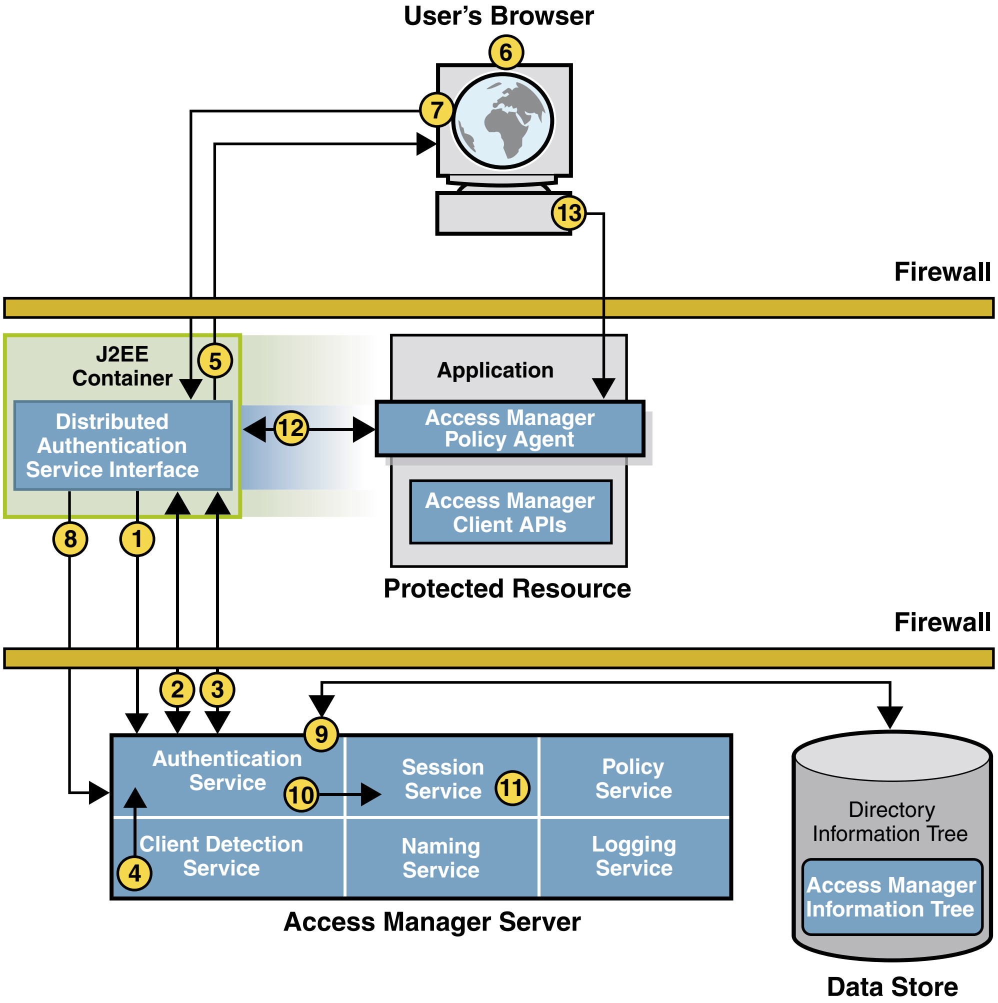
1. The user's browser sends an HTTP request to the protected resource.
2. The policy agent inspects the user's request, and no session token is found.
3. The policy agent contacts the configured authentication URL.

   In this example, the authentication URL it is set to the URL of the Distributed Authentication User Interface Service.
4. The browser sends a GET request to the Distributed Authentication User Interface.
5. The Session Service creates a new session, or data structure, and generates a session token. The session token is a randomly-generated string that represents the user.
6. Authentication Service sets the session data structure in a cookie.

The next part of the user session is User Authentication.

## User Authentication

When the browser sends a GET request to the Distributed Authentication User Interface, the events in the following illustration occur. The accompanying text describes the process.



1. Using the parameters in the GET request, the Distributed Authentication User Interface contacts the Authentication Service installed on the Access Manager Server.

2. Authentication Service determines the appropriate authentication module to use based upon Access Manager configuration and the request parameters passed by the Distributed Authentication User Interface through the Authentication client APIs.

   For example, if Access Manager is configured to use the LDAP Authentication type of module, the Authentication Service determines that the LDAP Authentication login page will be used.

3. Authentication Service determines which presentation callbacks should be presented, and sends to the Distributed Authentication User Interface all necessary credentials, requirements, and callbacks to be in used the presentation framework layer.

4. Client Detection Service determines which protocol, such as HTML or WML, to use to display the login page.

5. The Distributed Authentication User Interface returns to the Web browser a dynamic presentation extraction page along with the session cookie.

   The presentation extraction page contains the appropriate credentials request and callbacks info obtained from the Access Manager Server.

6. The user's browser displays the login page.

   The user enters information in the Username and Password fields of the login page.

7. The browser replies to the Distributed Authentication User Interface with a POST that contains the required credentials.

8. The Distributed Authentication User Interface uses the Authentication client APIs to pass credentials to the Access Manager Server.

9. The Authentication Service uses the appropriate authentication module type to validate the user's credentials.

   For example, if the LDAP authentication module type is used, Authentication Service verifies that the username and password provided exist in the LDAP directory. Other authentication module types have different requirements.

10. When authentication is successful, Authentication Service activates the session by calling the appropriate methods in the Session Service.

    Authentication Service stores information such as Login time, Authentication Scheme, and Authentication Level in the session data structure.

11. Once the session is activated, Session Service changes the state of the session token to valid.

12. The Distributed Authentication User Interface replies to the protected resource with an SSOToken in a set-cookie header.

13. The browser makes a request to the originally requested resource protected by an Agent.

    This time, the request includes the valid session data structure and session token that were created during the authentication process.

The next part of the user session is Session Validation.

## Session Validation

After authentication, when the user's browser redirects the initial HTTP request to the mail server for a second time, the events in the following illustration occur. The accompanying text describes the process.

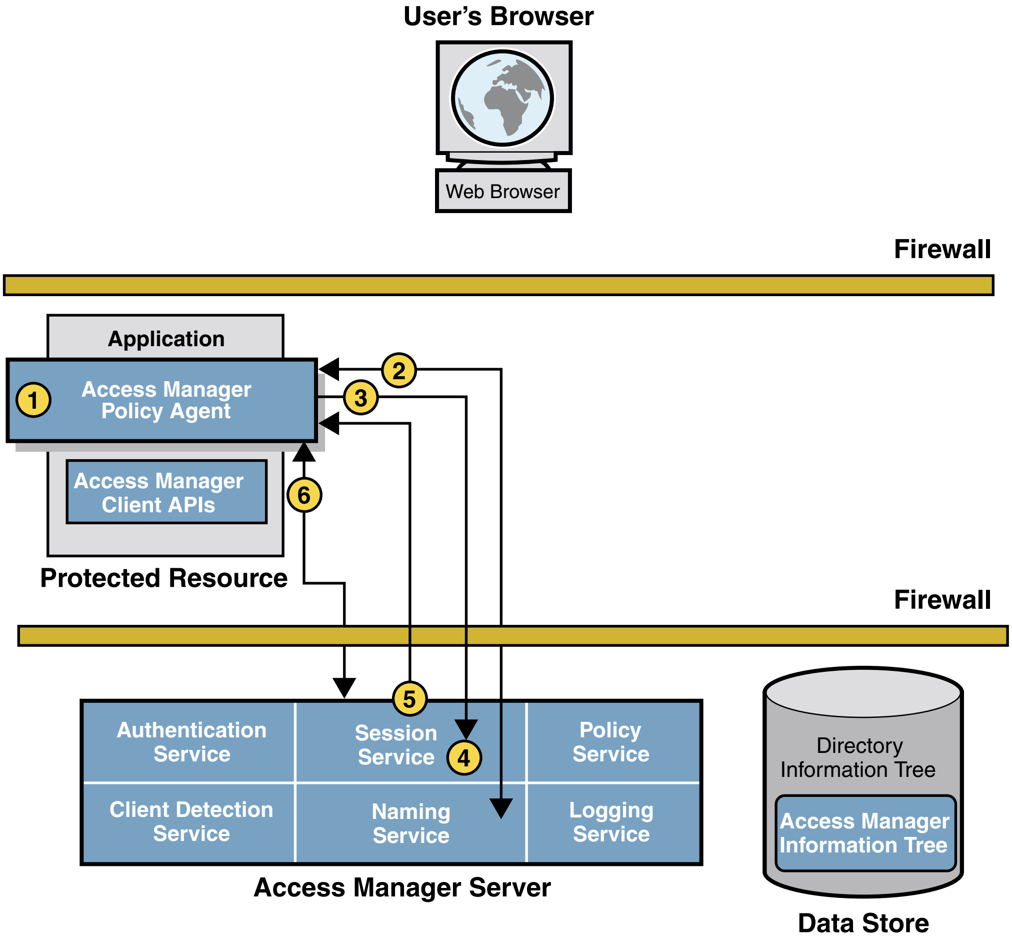


FIGURE 2–2 Session Validation

1. The policy agent intercepts the second access request.

   The request now contains a session token in the same DNS domain as Access Manager.
2. The policy agent determines the validity of the session token.

   a. The policy agent contacts the Naming Service to learn where the session token originated.

    The Naming Service allows clients to find the service URL for the internal services used by Access Manager. This information can then be used for communication regarding a session.

    b. The Naming Service decrypts the session token and returns the corresponding URLs . The URLs will 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 and 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 the Session Listener with the Session Service. This enables notification 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.

## Policy Evaluation and Enforcement

After a session token has been validated, the policy agent determines if the user can be granted access to the mail server by evaluating the defined policies. The following illustration and the text accompanying it describes the process.

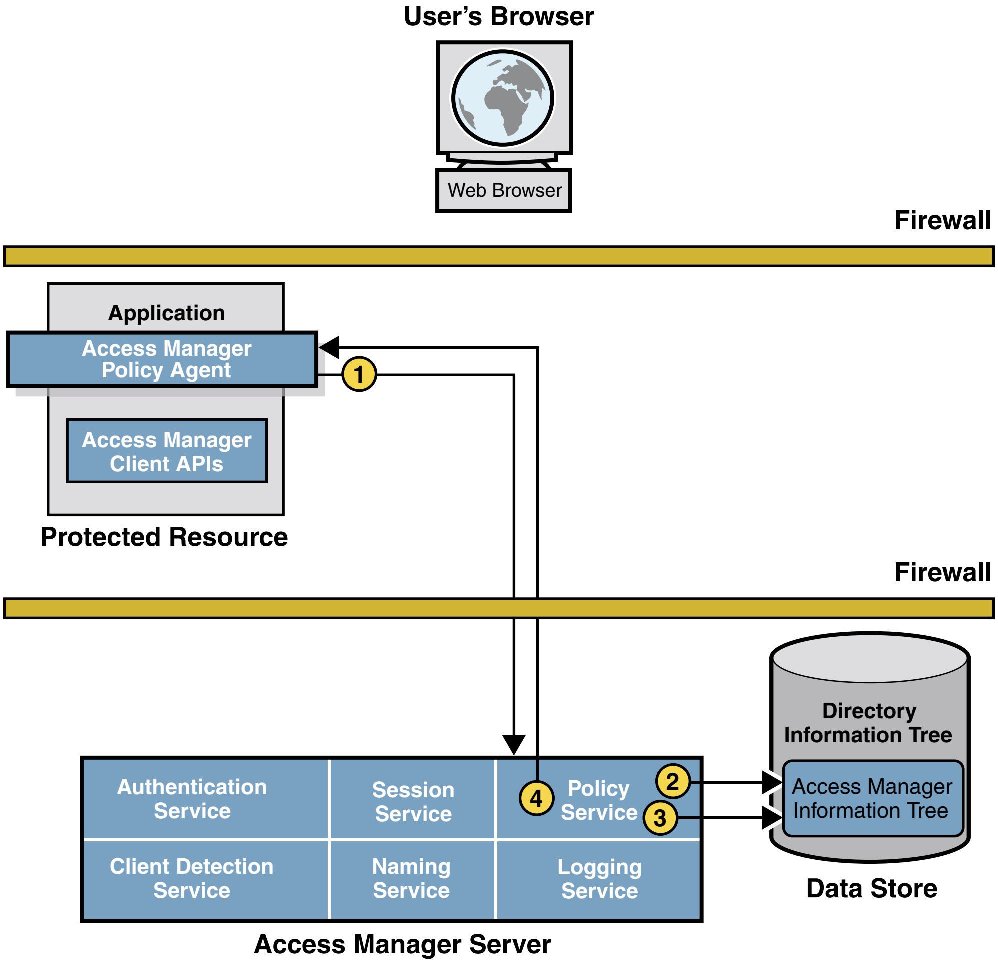


FIGURE 2–3 Policy Evaluation

1. The policy agent sends a request to the Policy Service.

   The request asks for decisions regarding resources in the policy agent's 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 Access Manager. If the policies have not been cached already, then the policies are loaded from the Access Manager information tree in the Identity Repository.

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 is denied access. Skip to step 5.

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

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

4. Policy service aggregates all policies that apply, and encodes a final decision to grant or deny access.

5. Policy Services responds to the policy agent with the appropriate decision.

   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.

The next part of the user session is logging the policy evaluation results.

## Results Logging

When the policy agent receives an allow decision from the Policy Service, the events in the following illustration occur. The accompanying text describes the process.

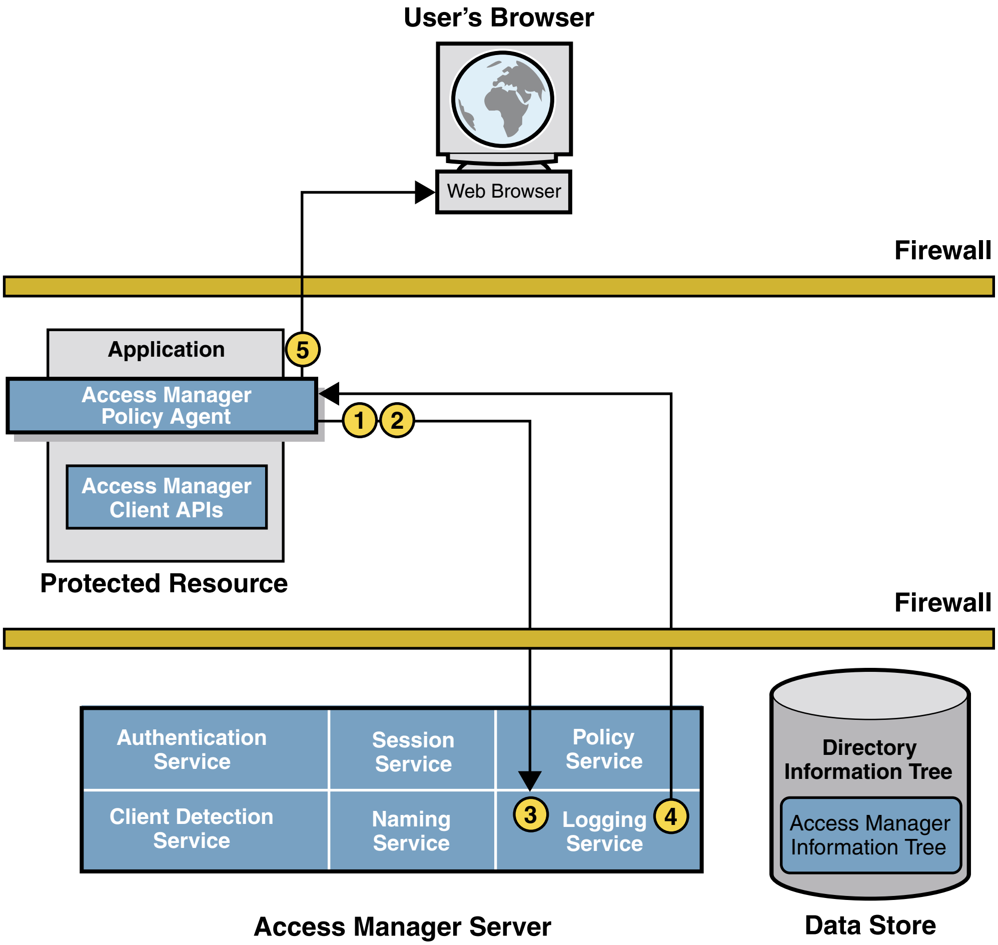


FIGURE 2–4 Logging the Policy Evaluation Results

1. The allow decision is cached in the policy agent, along with the session token, so that subsequent requests can be checked using the cache.

   It is no longer necessary for the policy agent to contact Access Manager. The cache will expire after an interval has passed or upon an explicit notification of change in policy or session status. The interval is configurable.
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.

   The browser displays the application interface. This basic user session is valid until it is terminated. See "Session Termination" on page 45.

   While the user is still logged in, if he attempts to log into another protected resource, then the Single Sign-On session begins.

# Single Sign-On Session

SSO is always preceded by a basic user session in which a session is created and its session token is validated, and in which the user is authenticated. For detailed information, see "Basic User Session" on page 32.

SSO begins when the authenticated user requests a protected resource on a second server in the same DNS domain. The following example describes an SSO session by tracing what happens when an authenticated user (from "Basic User Session" on page 32) accesses a second application, an expense reporting application. Session Service maintains user session information with input from all applications participating in the single sign-on. In this example, session service maintains information from the benefits administration and the expense reporting application. The following events occur.

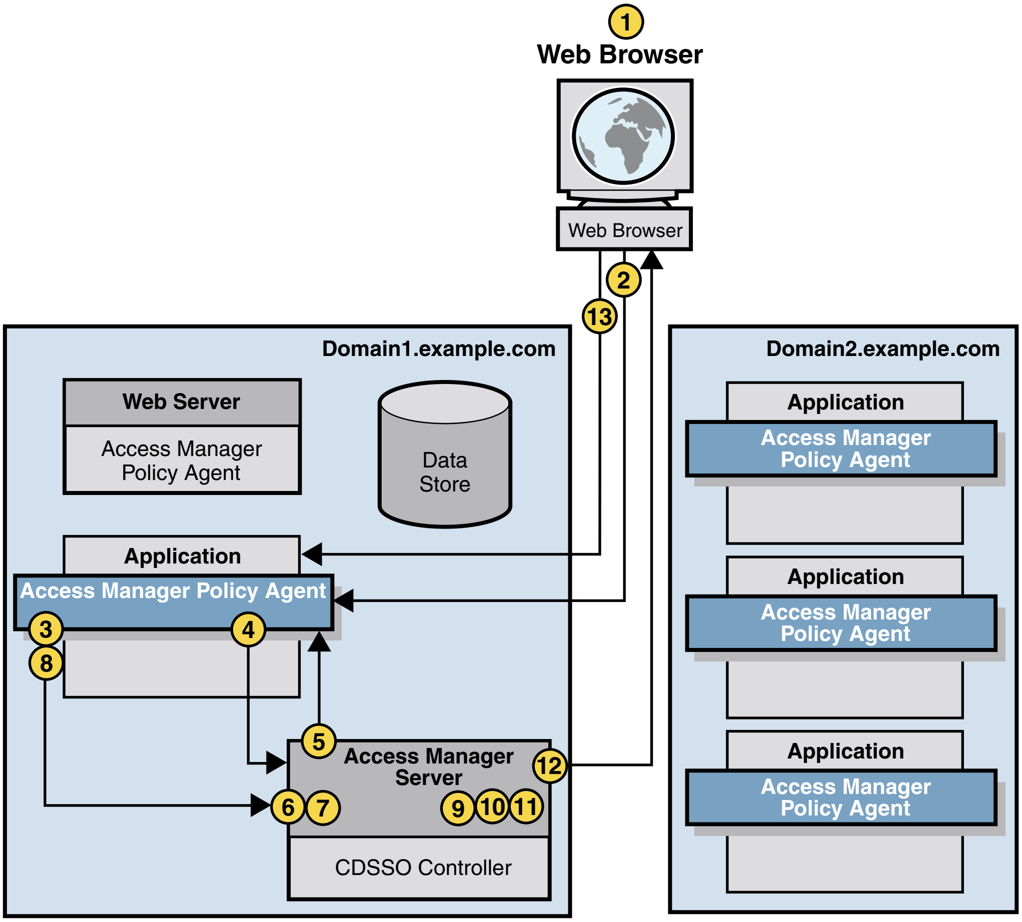


FIGURE 2–5 Single Sign-On Session

1. The user attempts to access an expense reporting application.

   Both the expense reporting application and the benefits administration application from section "Basic User Session" on page 32 are hosted by servers in the same domain.

2. The user's browser sends An HTTP request to the expense reporting application. The request includes the user's session token.

3. The policy agent intercepts and inspects the request to determine whether a session token exists.

   A session token indicates the user is already authenticated. The user was authenticated when the user logged in to the benefits administration application, so authentication service is not required at this time. The SSO APIs retrieve the session data structure, which is known to SSO APIs as the *SSOToken*. The session token, or session ID, is known to SSO APIs as the *SSOTokenID*.

4. The policy agent determines the validity of the session.

   For detailed steps, see "Session Validation" on page 36.

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

   A Session Listener allows notification to the policy agent when a change in the SSOToken state or validity occurs.
8. The policy agent sends a request to the Policy Service.

   The request asks for a decision regarding resources in the policy agent's portion of the HTTP namespace.
9. The Policy Service checks for policies that apply to the request.
10. If Policy Service does not find policy allowing access to the protected resource, the user is denied access. The following events occur:
    a. The Logging Service logs this denial of access.
    b. The policy agent issues a *Forbidden* message to the user.

       The user can then be redirected to an administrator-specified page indicating the user was denied access.
11. If Policy Service finds policy allowing access to the protected resource, the user is granted access to the protected resource.

    The SSO session is valid until it is terminated. "Session Termination" on page 45.

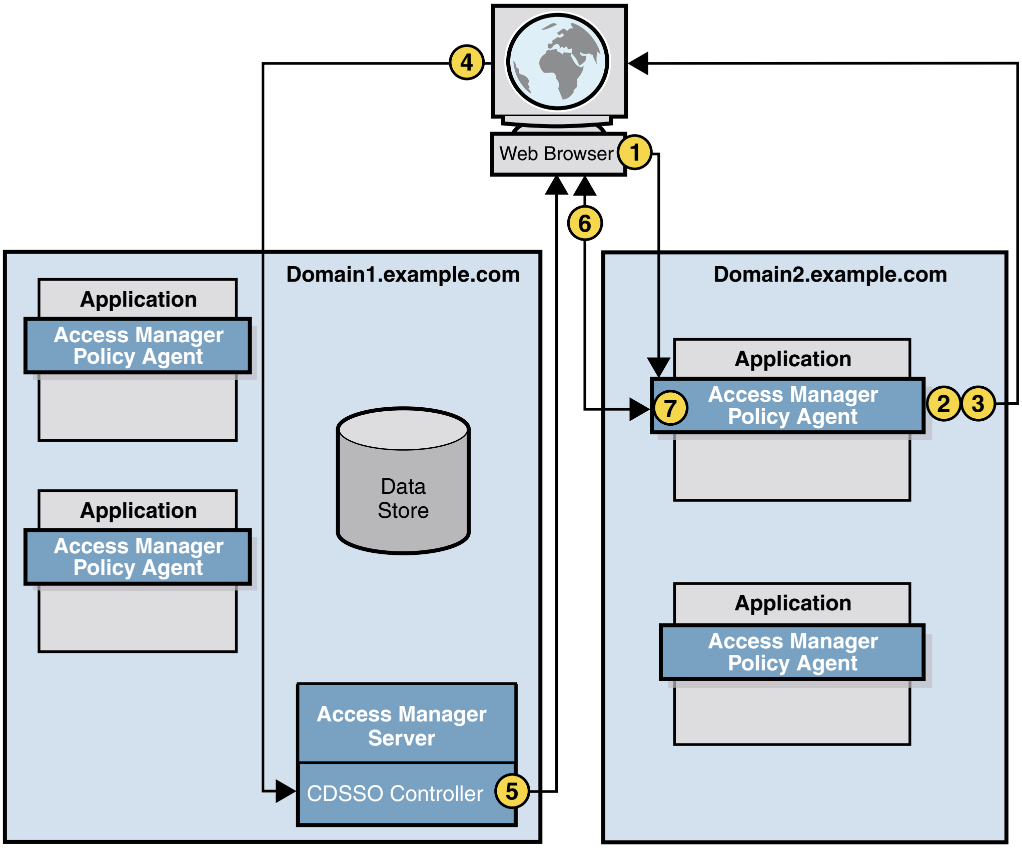
    While the user is still logged in, if the user decides to attempt to log in to another protected resource located in a different DNS domain, then CDSSO takes place.

# Cross-Domain Single Sign-On Session

CDSSO occurs when an authenticated user requests a protected resource on a different server in a different DNS domain. The user in the previous sections, "Basic User Session" on page 32 and "Single Sign-On Session" on page 41, for example, accessed applications in his company's DNS domain. In the following example, the same user will log in to a benefits administration application supplied to his company by an external company. The benefits administration application is hosted on the external company's DNS domain.

The CDSSO servlet within Access Manager transfers the user's SSOToken into the external DNS domain, making the SSOToken usable for applications in this second domain. The CDSSO Controller Service uses Liberty Protocols to transfer sessions between domains.

When the user logs in to the benefits administration application in the external DNS domain, the events in the following illustration occur. The process is described in the accompanying text.



1. The user's browser sends an HTTP request to the benefits administration application.

2. The policy agent intercepts the request and inspects it to determine if a session token or SSOToken exists for the *current* domain.

   If a session token or SSOToken is present, the policy agent validates the session. If no session token or SSOToken is present, the policy agent sends the HTTP request to the CDSSO Controller Service (including the relevant Liberty Alliance Project parameters). In this example, we assume a token is found.

3. The policy agent finds a session token and validates the session.

   For detailed information, see "Session Validation" on page 36.

4. The user's browser allows the redirect.

   This time the request contains the SSOToken. Recall that earlier in the user session (see "Single Sign-On Session" on page 41), the SSOToken was set in a cookie in the primary domain.

5. The CDC Servlet receives the SSOToken, and replies to the browser with a Liberty Post Profile response that includes user session information.

   a. The user's browser automatically submits the form containing the Liberty document to the policy agent.

      The form is based upon the Action and the Javascript included in the Body tags `onLoad`.

   b. The policy agent receives the Liberty document and extracts the user session information.

6. The policy agent validates the SSOtoken.

   For detailed information, see "Session Validation" on page 36.

7. The policy agent sets the SSOToken in a cookie for the new DNS domain.

   The CDSSO session is now valid until it is terminated, and the cookie can be used by all agents in the new domain.

8. The policy service examines policies.

   For detailed information, see "Policy Evaluation and Enforcement" on page 38.

9. The policy agent allows or denies access to the requested resource.

   For detailed steps, see "Results Logging" on page 40. In this case, the session token was determined to be valid, and the user is allowed access.

10. The policy agent responds to the user by presenting the benefits administration application screen.

# Session Termination

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

- "User Ends Session" on page 46
- "Administrator Ends Session" on page 46
- "Access Manager Enforces Timeout Rules" on page 46
- "Session Quota Constraints" on page 47

## User Ends Session

When a user explicitly logs out of Access Manager the following events occur:

1. The user logs out by clicking on a link to the Logout Service.
2. The Logout Service receives the Logout request, and then performs the following steps:
   a. Marks user's session as destroyed.
   b. Destroys session.
   c. Returns a successful logout page to the user.
3. 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.
4. The policy agents flush the session from cache and the user session ends.

## Administrator Ends Session

Access Manager Administrators with appropriate permissions can terminate a user session at any time. When an administrator ends a session, the following events occur:

1. The administrator uses Sessions tab in the Access Manager console to end the user's session.
2. The Logout Service receives the Logout request, and then performs the following steps:
   a. Marks user's session as destroyed.
   b. Destroys session.
3. 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.
4. The policy agents flush the session from cache and the user session ends.

## Access Manager Enforces Timeout Rules

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

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

## Session Quota Constraints

Access Manager 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 3

# User Authentication

Access Manager 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 Authentication service works with other Access Manager components to authenticate a user, or prove that the user's identity is genuine.

Topics in this chapter include:

## Authentication Overview

The following example demonstrates a user's view of how Authentication service works. A company employee must look up a colleague's phone number, so he uses a browser to access the company's online phone book. To log in to the phone book service, the employee must provide a user name and password. Access Manager compares the user's input with data stored in Directory Server. If Access Manager finds a match for the user name, and if the given password matches the password stored in Directory Server, Access Manager authenticates the user's identity. After authentication, the policy evaluation process occurs. If the policy agent allows access to the user, and the corporate phone book is displayed.

See "Basic User Session" on page 32 for a detailed description and illustration of the authentication process within a basic user session.

Authentication Service is client-type aware and supports all configured client types such as cookieless and cookie-enabled client types.

# Authentication Plug-In Modules

An authentication module is a plug-in that collects user information such as a user ID and password, and then checks the information against entries in a database. If a user provides information that meets the authentication criteria, then the user is granted access to the requested resource. If the user provides information that does not meet authentication criteria, the user is denied access to the requested resource. Access Manager is installed with 15 types of authentication modules. The following table provides a brief description of the 15 default authentication module types.

TABLE 3–1 Access Manage Authentication Module Types

| Authentication Module Name | Description |
|---|---|
| Active Directory | Uses an Active Directory operation to associates a user ID 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 | Enabless 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 Access Manager 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 module can require the use of the Online Certificate Status Protocol (OCSP) to determine the state of a certificate. Use of the OCSP is optional. The user is granted or denied access to a resource based on whether or not the certificate is valid. |
| Data Store | Enables authentication against one or more configuration data stores within a realm. |
| HTTP Basic | Enables authentication to occur with no data encryption. Credentials are validated internally using the LDAP 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 ID password with a particular LDAP entry. You can define multiple LDAP authentication configurations for a realm. |

TABLE 3–1 Access Manage Authentication Module Types *(Continued)*

| Authentication Module Name | Description |
|---|---|
| Membership | Enables user to self-register. The user create 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 Directory Server 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. |
| Windows Desktop Single Sign-On (SSO) | Also known as Kerebos authentication, this module is specific only to the Windows operating system. Allows a user who has already authenticated with a key distribution center to be authenticated with Access Manager without having to provide the login information again. |
| Windows NT | Uses a Microsoft Windows NT™ server to verify identities. |

After granting or denying access, Access Manager checks for information about where to redirect the user. Access Manager uses a specific order of precedence when checking 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. Five types of authentication exist including Realm-based and Role-based authentication. See "Authentication Type Configurations" on page 54 for more information about authentication types.

You can use the Access Manager Console to enable and configure authentication module types that come with Access Manager by default. You can also create and configure multiple instances of a particular authentication module type. An instance is a child entity that extends the schema of a parent authentication module and adds its own subschema. See *Sun Java System Access Manager 7.1 Administration Guide* for detailed information about enabling and configuring default authentication modules types and authentication module instances.

You can also write your own custom authentication module or plug-in to connect to the Access Manager authentication framework. For more information about writing custom authentication modules, see the *Sun Java System Access Manager 7.1 Developer's Guide*.

# Authentication Framework

The Authentication framework includes two pluggable and customizable services: General Authentication Service, and Authentication Configuration Service.

## General Authentication Service

The general authentication service is used for server-related attribute configuration. Some of the attributes described in this service are default attributes for all Access Manager authentication modules.

You must register the general authentication service as a service to a realm before a user can use authentication modules to log in. The general authentication service enables the Access Manager administrator to define default values for a realm's authentication parameters. These values can be used if no overriding value is defined in the specified authentication module. The default values for the General Authentication Service are defined in the `amAuth.xml` file and stored in the Access Manager information tree after installation.

## Authentication Configuration Service

The Authentication Configuration Service describes all the dynamic attributes for service-based authentication. This service is used for 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.

# Distributed Authentication User Interface Component

The Distributed Authentication user interface enables a policy agent or an application that is deployed in a non-secured area to communicate with the Access Manager Authentication Service that is installed in a secured area of the deployment. Typically, the non-secured policy agent or application is separated from Access Manager by two firewalls. In such deployments, policy agents and applications are not usually allowed to communicate across two firewalls.

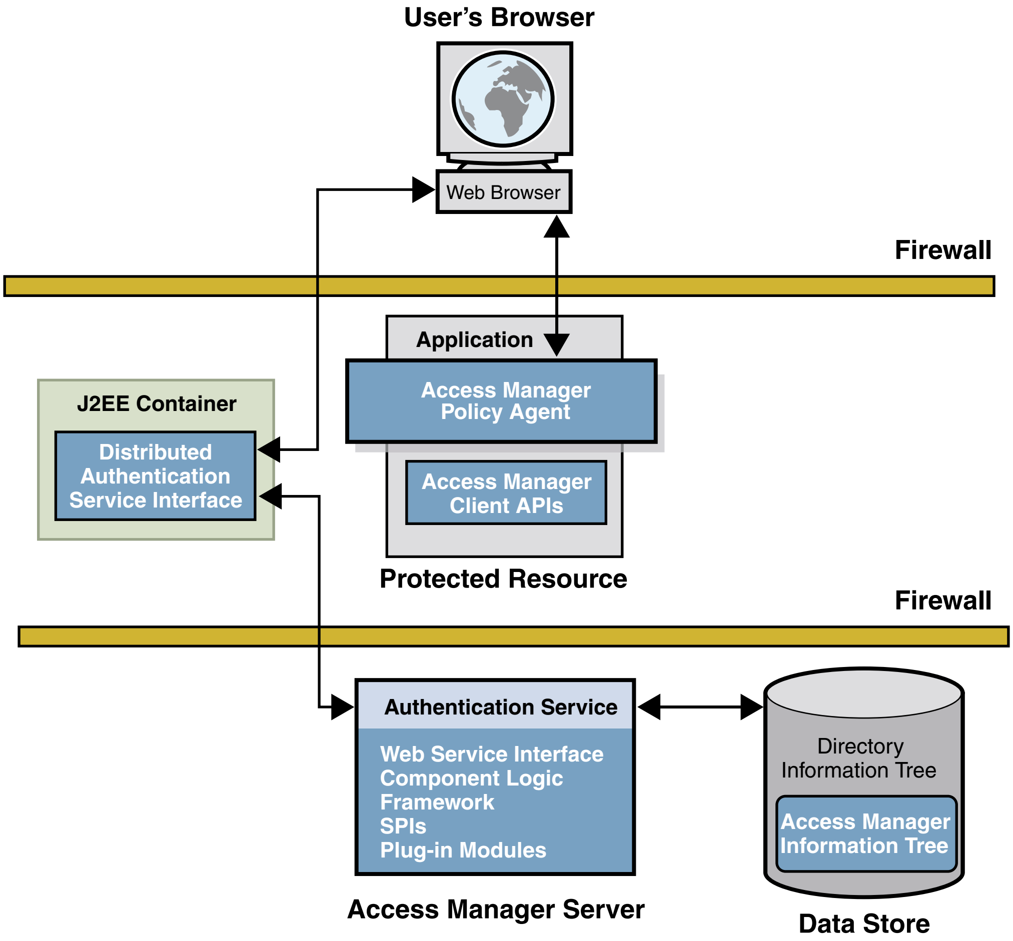


**FIGURE 3–1** Distributed Authentication

Access Manager provides a remote Authentication user interface component to enable secure, distributed authentication across two firewalls. You can install the remote authentication user interface component on any servlet-compliant web container within the non—secure layer of an

Access Manager deployment. The remote component works with Authentication client APIs and authentication utility classes to authenticate web users.

The remote component is customizable and uses a JATO presentation framework. You can install the distributed authentication user interface on a J2EE web container within the non—secure layer of an Access Manager deployment. The web browser communicates an HTTP request to the remote authentication user interface, which in turn presents a login page to the user. The web browser sends user login information through a firewall to the remote authentication user interface. The remote authentication user interface communicates through the second firewall to the Access Manager Server. For detailed illustration and process flow, see "User Authentication" on page 34. For detailed installation and configuration instructions, see the *Sun Java System Access Manager 7.1 Administration Guide*.

# Inside the Core Authentication Component

The core Authentication component is where default configurations are stored and where authentication processes are invoked.

## Client Detection

An initial step in the authenticating process is to identify the type of client making the HTTP(S) request. This Access Manager 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 Netscape browser is used to request a web page, Access Manager displays an HTML login page. Once the user is validated, the client type `Netscape browser` is added to the session token.

## Authentication Type Configurations

When you install Access Manager, a number of authentication types are automatically configured for you. The following types of authentication are available to you by default when you install Access Manager.

| | |
|---|---|
| Realm-based Authentication. | User authenticates to a realm or subrealm in the Access Manager information tree. |
| Role-based Authentication. | User authenticates to a role within a realm or subrealm of the directory information tree. A role is a group of like items in the directory. A static role is created when an attribute is assigned to a specific user or container in the directory. A filtered role is dynamically generated based on an attribute contained in the a user's or container's LDAP entry. |

| | |
|---|---|
| | For example, all users that contain a particular attribute, for example employee, can be automatically included in a filtered role named employees. |
| Service-based Authentication. | User authenticates to a specific service or application registered to a realm or subrealm. |
| User-based Authentication. | User authenticates using an authentication process configured specifically for him or her. |
| Authentication Level-based Authentication | Administrator specifies the security level of the modules to which identities can authenticate. |
| Module-based Authentication. | User specifies the module instance to which the user will authenticate. |

## Redirection URLs

In the last phase of the authentication process, Access Manager either grants or denies access to the user. If access is granted, Access Manager uses a login URL to display a login page in the browser. If access is denied, Access Manager 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 uses a login URL or redirection URL depending on a specific order of precedence. The order of precedence is based on the authentication type used (realm-based, role-based, and so forth), and on whether authentication succeeded or failed. For a detailed description of how Access Manager proceeds through the order of precedence, see Chapter 4, “Managing Authentication,” in *Sun Java System Access Manager 7.1 Administration Guide*.

## Account Locking

The Authentication Service provides an account locking feature that “locks out” or prevents a user from completing the authentication process after a specified number of failures. Only modules that throw an Invalid Password Exception can leverage the Account Locking feature. Access Manager sends email notifications to administrators when account lockouts occur. Account locking activities are also logged. The account locking feature is disabled by default. You can enable account locking by using the Access Manager console.

Access Manager supports two types of account locking: Physical Locking and Memory Locking.

Physical Locking. By default, user accounts are `active` or physically unlocked. You can initiate physical locking by changing the status of an LDAP attribute in the user’s profile to `inactive`. The account remains physically locked until the attribute is changed to `active`.

Memory Locking. You can enable memory locking by changing the Login Failure Lockout Duration attribute to a value greater then `0`. The user's account is locked in memory for the number of minutes you specified. The account is unlocked after the time period elapses. You can configure Memory Locking so that a user account is locked in memory after a specified number of tries. The user account will be locked when AM server is restarted.

## Authentication Chaining

You can configure one or more authentication module instances so that a user must pass authentication credentials to all authentication modules instances before the user is allowed access. This feature is called as authentication chaining. Determining access is based upon control flags you specify for the chain. Access Manager uses the Java Authentication and Authorization Service (JAAS) framework to implement authentication chaining. The JAAS framework is integrated in the Authentication Service.

You can configure authentication chaining by realm, user, role, or service configuration. Authentication modules use a control flags to indicate requirements for successful authentication.

Each registered authentication module type is assigned one of the following control flags:

Requisite. The LoginModule is required to succeed. If it succeeds, authentication continues down the LoginModule list. If it fails, control immediately returns to the application (authentication does not proceed down the LoginModule list).

Required. Authentication to this module is required to succeed. If any of the required modules in the chain fails, the whole authentication chain will fail and the user will be notified of this.

Sufficient. The LoginModule is not required to succeed. If it does succeed, control immediately returns to the application (authentication does not proceed down the LoginModule list). If it fails, authentication continues down the LoginModule list.

Optional. The LoginModule is not required to succeed. Whether it succeeds or fails, authentication still continues to proceed down the LoginModule list.

Once authentication to all modules in the chain is successful, control is returned to the Authentication Service from the JAAS framework. The JAAS framework validates all user IDs used during the authentication process, and then maps them all to one user. The mapping is based on the configuration of the User Alias List attribute in the user's profile.

If all the maps are correct, then a valid session token is issued to the user. If all the maps are not correct, then the user is denied a valid session token.

## 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 is also used to allow access to one instance of Access Manager using many different aliases. For example, you might configure one instance of Access Manager as `intranet.example.com` for employees and `extranet.example.com` for partners.

## Persistent Cookie

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.

## Session Upgrade

The Authentication Service enables for the upgrade of a valid session token based on a second, successful authentication performed by the same user to one realm. If a user with a valid session token attempts to authenticate to a resource secured by his current realm, and this second authentication request is successful, 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 by 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.

## Validation Plug-in Interface

The Validation Plug-In Interface is supported by only the LDAP and Membership authentication module types. An administrator can write username or password validation logic appropriate for a particular realm, and then the plug logic into the Authentication Service. Before authenticating a user or changing the user password, Access Manager will invokes this plug-in. If the validation is successful, authentication continues. If validation fails, an authentication failed page will be thrown. The plug-in extends the `com.iplanet.am.sdk.AMUserPasswordValidation` class which is part of the Service Configuration SPI.

## JAAS Shared State

The JAAS shared state enable sharing of both user ID and 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.

# Presentation Layer

The Authentication Service implements a user interface that is separate from the Access Manager 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 user name and password. Once the credentials have been passed back to Access Manager and authentication is successful, the user can gain access based on the user's specific privileges:

- Administrators can access the administration portion of the Access Manager 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.

Access Manager 7.1 provides customization support for the Authentication Service user interface. You can customize JavaServer Pages™ (JSP™) and the file directory level for `/org/service/locale/client_type.` See the *Sun Java System Access Manager 7.1 Developer's Guide* for more information.

# Distributed Authentication User Interface

Access Manager provides a remote Authentication user interface component to enable secure, distributed authentication across two firewalls. You can install the remote authentication user interface component on any servlet-compliant web container within the non—secure layer of an Access Manager deployment. The remote component works with Authentication client APIs and authentication utility classes to authenticate web users. The remote component is customizable and uses a JATO presentation framework.

# Authentication Programming Interfaces

Access Manager provides both Java APIs and C APIs for writing authentication clients that remote applications can use to gain access to the Authenticate Service. Both Java and C APIs support all Authentication types supported by Web User Interface. This communication between the APIs and the Authentication Service occurs by sending XML messages over HTTP(S). Clients other than Java and C clients can user the XML/HTTP interface directly to initiate an Authentication request.

You can add Authentication modules to Access Manager by using the `com.iplanet.authentication.spi` package. The 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 about using Authentication programming interfaces, see the *Sun Java System Access Manager 7.1 Developer's Guide*.

CHAPTER 4

# Authorization and the Policy Service

Access Manager Policy Service determines if a user has been given permission by a recognized authority to access a protected resource. This process is known as user authorization. This chapter describes how the various parts of the Policy Service work together to perform authorization. Topics covered in this chapter include:

## Policy Framework

The policy framework in Access Manager is where policy management logic and evaluation logic are implemented. The framework consists of a general Policy Service and the Policy Configuration Service.

The general Policy Service performs three main functions:

- Provides a means for defining and managing access policies.
- Evaluates access policies.
- Acts as a policy decision point (PDP) to deliver the result of the policy evaluation.

Applications host resources. In Access Manager, applications are protected by policy enforcement points (PEP) such as J2EE or web policy agents to enforce access control. Access control is based on the policy decision provided by policy evaluation at the PDP which is the Policy Service.

A policy is a rule that describes who is authorized to access a resource. Policies are grouped into access control realms which together form the Access Manager information tree.

When a user attempts to access a resource protected by a PEP, the PEP talks to the PDP to get a policy decision. At the PDP, which is Access Manager, Policy Service determines and evaluates policies that protect the resource and are applicable to the user. This results in a policy decision indicating whether the user is allowed to access the resource. Upon receiving the decision, the PEP allows or denies access appropriately. This whole process is called authorization.

The general Policy Service enables an administrator to configure custom policy plug-ins by providing names and class location of the custom plug-ins. The Policy Configuration Service provides a means to specify how policies will be defined and evaluated within a realm or subrealm. The Policy Configuration services enables you to specify: which directory to use for subject lookup; which search filters to use; which subjects, conditions, and response providers to use.

# Access Control Realms

You create an access control realm when you want to apply policies to a group of related services or servers. An Access Manager realm is a group of authentication and authorization properties that you can associate with a user or group of users, or a collection of protected resources. For example, you can create a realm that groups all servers and services that are accessed regularly by your employees in one region. Within that regional grouping or realm, you can group all servers and services accessed regularly by employees in a specific division such as Human Resources. For example, a policy might state that all Human Resources administrators can access the URL `http://HR.example.com/HRadmins/index.html`. You might add constraints to this policy. For example: The policy is applicable only Monday through Friday from 9:00 a.m. through 5:00 p.m.

Realm data is stored in an Access Manager information tree. Realms facilitate the delegation of policy management privileges within a realm hierarchy.

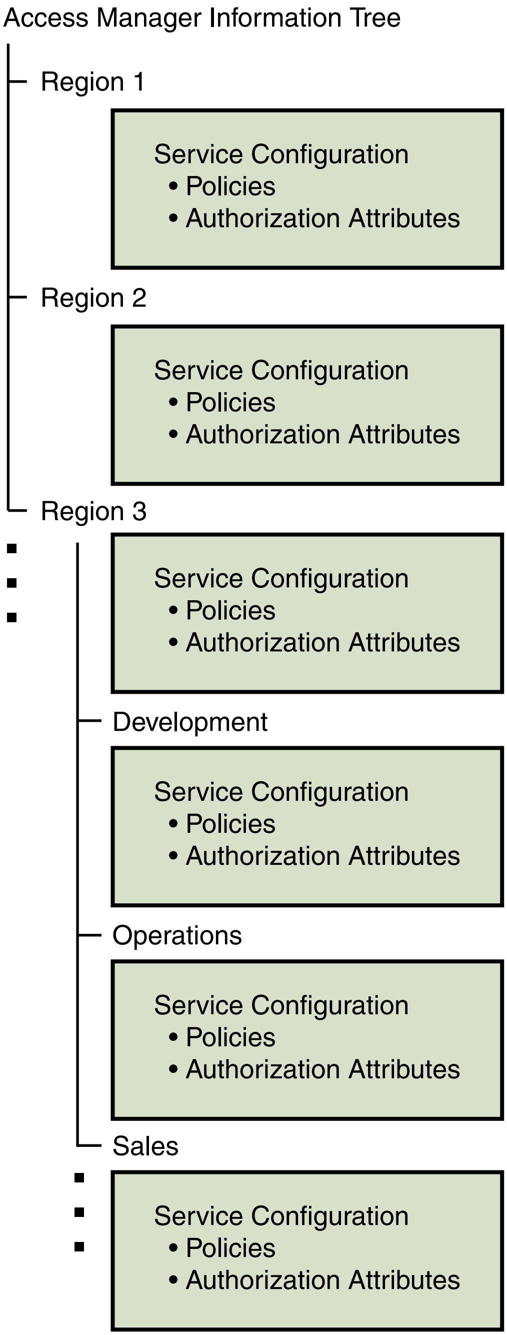


**FIGURE 4–1** Access Manager Information Tree

## Access Manager Information Tree

Access Manager creates a special and proprietary branch in a data store such as an LDAP directory for storing realm configurations, authentication properties, and authorization policies. This directory can be different from the directory hosting the Access Manager Identity Repository. Together the realms form the Access Manager information tree. The Access Manager information tree is separate from the user branch in the Identity Repository.

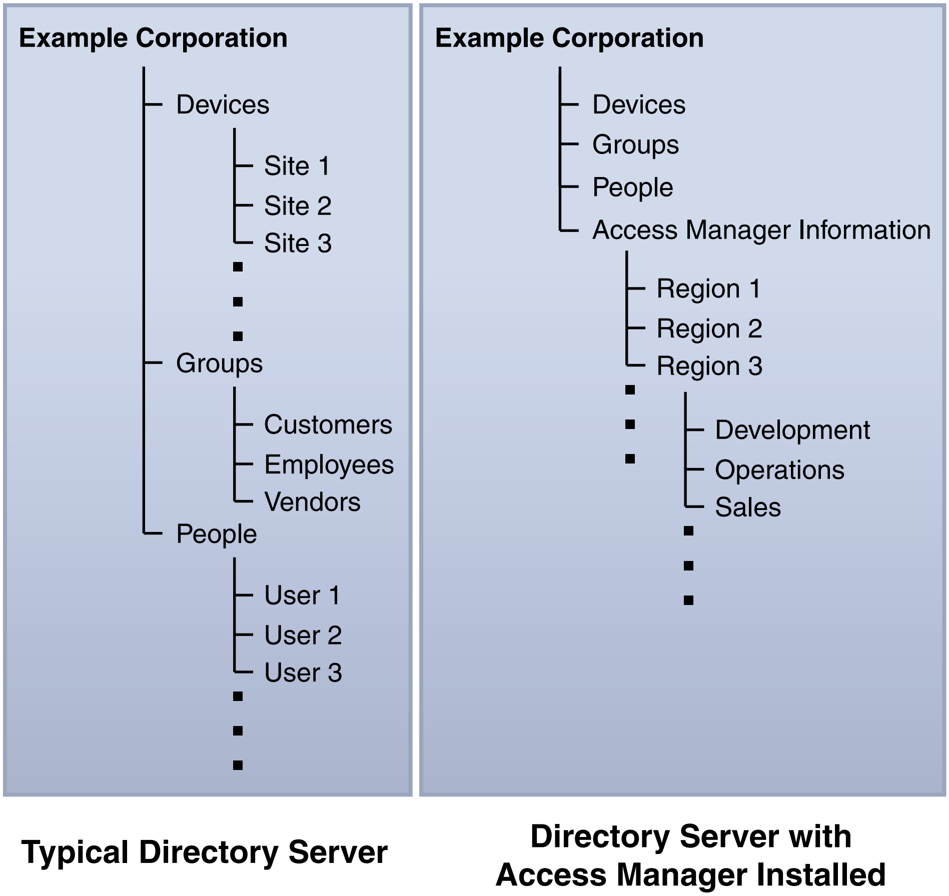


FIGURE 4–2 Access Manager Information Tree Within an Identity Repository

Access Manager components and plug-ins access the data stored in the Access Manager information tree, and use data for various purposes. The following are some examples:

- Policy runtime accesses policy data for policy evaluation.
- Identity Repository plug-in finds configuration information for data stores.
- Authentication Service finds authentication configuration information.

# About Authorization Policies

The Policy Service authorizes a user based on the policies stored in the access control information tree. You can create two types of Access Manager policies: normal policies and referral policies. Create a normal policy when you want to define access privileges for a resource. Create a referral policy when you want to delegate policy creation to another entity such as a peer realm, a subrealm, or a third-party product.

## Normal Policy

A normal policy specifies a protected resource and also specifies who is allowed to access the resource. The protected resource can be anything hosted by a protected server. Examples of protected resources are applications, content such as document files, or the server itself. Only a Top-Level Realm or Policy Administrator can create or manage polices that apply to any resource.

A normal policy consists of rules, subjects, conditions, and response providers.

### Policy Rules

A rule defines a policy 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 website, 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`.

### Policy Subjects

A subject specifies by implication the user or collection of users that the policy affects. You can implement custom subjects by using Policy APIs. You can assign subjects to policies. Access Manager includes the following subjects:

Access Manger Roles
The roles you create and manage under the Realms Subject tab can be added as a value of the subject.

Access Manager Identity
The identities you create and manage under the Realms Subject tab can be added as a value of the subject.

Authenticated Users
Any user with a valid SSOToken is a member of this subject. All authenticated users would be member of this Subject, even if they have authenticated to a realm that is different from the realm in which the policy is defined. This is useful if the resource owner would like to give access to resources that is managed for users from other realms.

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 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 organization can be added as a value of this subject

Web Services Clients

Valid values are the DNs of trusted certificates in the local JKS keystore, which correspond to the certificates of trusted WSCs. This subject has dependency on the Liberty Web Services Framework and should be used only by Liberty Service Providers to authorize WSCs. A web service client (WSC) identified by the SSOToken is a member of this subject, if the DN of any principal contained in the SSOToken matches any selected value of this subject.

## Policy 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. You can implement custom conditions using the Policy APIs. Access Manager provides the following conditions:

Authentication Level

The policy applies if the user's authentication level is greater than or equal to the Authentication level set in the condition. The Authentication Level attribute indicates the level of trust for authentication.

Authentication Scheme

Policy is applicable based on which authentication scheme is specified.

IP Address

Policy is applicable based on a range of IP Addresses.

LE Authentication Level

Policy is applicable if the user's authentication level is less than or equal to the Authentication level set in the condition.

Session

Policy is applicable based on user session data such as Max Session Time.

Session Property

Policy is applicable based on values of properties set in the user's Access Manager session.

Time

Policy is applicable based on time constraints.

## Policy Response Providers

Response providers are plug-ins that provide policy-based response attributes. The response provider attributes are sent with policy decisions to the PEP. Access Manager includes one implementation, the `IDResponseProvider`. Custom response providers are not supported in this version of Access Manager. Agents, PEPs, typically pass these response attributes as headers to applications. Applications typically use these attributes to personalize application pages such as a portal page.

# Referral Policy

A referral policy enables a Realm Administrator or a Policy Administrator to delegate policy configuration tasks. A Realm Administrator or Policy Administrator at the root or top level of the Access Manager information tree can create policy for any resource. An administrator or Policy Administrator for realms below the top level have permissions to create policies for only resources delegated to the realm. The Realm Administrator or Policy Administrator can use referral policies to delegate policy management privileges for a collection of resources to other realms.

You can implement custom referrals by using the Policy APIs. Access Manager provides the following referrals:

| | |
|---|---|
| Peer Realm Referral | Administrator can delegate policy management privileges to a peer realm. |
| Subrealm Referral | Administrator can delegate policy management privileges to a subrealm. |

A referral policy delegates both policy creation and policy evaluation. A referral policy consists of one or more rules and one or more referrals.

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

For example, a top-level realm exists named ISP. It contains two subrealms named company1 and company2. The Top-Level Administrator for ISP wants to 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 company 2 real. The Top-Level Administrator creates two referral policies:

- Referral Policy 1

  Resource Name: `http://company1.com`

  Subrealm Referral Value: `company1`
- Referral Policy 2

  Resource Name: `http://company2.com`

  Subrealm Referral Value : `company2`

# Policy SPIs and Plug-Ins Layer

Access Manager includes SPIs that work with the Policy framework to create and manage policies. You can develop customized plug-ins for creating custom policy subjects, referrals, conditions, and response providers. For information on creating custom policy plug-ins, see the *Sun Java System Access Manager 7.1 Developer's Guide*.

The following table summarizes the Policy SPIs , and lists the specialized Policy plug-ins that come bundled with Access Manager.

**TABLE 4–1** Policy Service Provider Interfaces (SPIs)

| Interface | Description |
|---|---|
| Subject | Defines a set of authenticated users for whom policy applies.<br>The following Subject plug-ins come bundled with Access Manager: Access Manager Identity Subject, Access Manager Roles, Authenticated Users, LDAP Groups, LDAP Roles, LDAP Users, Organization Web, and Services Clients. |
| Referral | Delegates management of policy definitions to another access control realm. |
| Condition | Specifies applicability of policy based on conditions such as IP address, time of day, authentication level.<br>The following Condition plug-ins come bundled with Access Manager: Authentication Level, Authentication Scheme, IP Address, LE Authentication Level, Session, SessionProperty, and Time. |
| Resource Name | Allows a pluggable resource. |
| Response Provider | Gets attributes that are sent along with policy decision to the policy agent, and used by the policy agent to customize the client applications. Custom implementations of this interface are now supported in Access Manager 7.1. |

# Policy Client APIs

Access Manager provides client APIs that implement policy evaluation logic on a remote web server or application server. For policy client API information, see the *Sun Java System Access Manager 7.1 Developer's Guide*.

CHAPTER 5

# Federation Management, SAML, and Web Services

This chapter explains the concept of identity federation, and describes the role of the Federation Management feature in Access Manager. For detailed information about enabling or managing identity federation, or using the Federation Management APIs and SPIs, see the *Sun Java System Access Manager 7.1 Federation and SAML Administration Guide*.

This chapter includes the following topics:

- "The Need for Federated Identities" on page 69
- "The Liberty Alliance Project" on page 70
- "Federation Management Implemented in Access Manager" on page 73
- "Federation Management Protocols Flow" on page 78

## The Need for Federated Identities

Consider the many times an individual accesses services on the Internet in a single day. At work, he uses the company intranet to perform a multitude of business-related tasks such as reading and sending email, looking up information in the company phone book and other internal databases, and submitting expense reports and other business-related online forms. At home after work, he checks his personal email, then logs into an online news service to check his baseball team's standings. He may finalize his travel plans via his travel agent's website, and then does some online shopping at his favorite clothing store. Each time he accesses a service on the Internet, he must log in and identify himself to the service provider.

A *local identity* refers to the set of attributes or information that identify a user to a particular service provider. These attributes typically include a name and password, plus an email address, account number or other identifier. For example, the individual in our scenario is known to his company's network as an employee number, but he is known to his travel agent as Joe Smith. He is known as an account number to the car rental agency he uses frequently. He is known to his favorite airline by a different account number. He uses one email name and address for his personal email, and a different email name and address for his workplace. Each of these different user names represents a different local identity.

Identity federation allows a user to consolidate the many local identities he has configured among multiple service providers. With one *federated identity*, the individual can log in at one service provider's site and move to an affiliated service provider site without having to re-authenticate or re-establish his identity. For example, with a federated identity, the individual might want to access both his personal email account and his business email account from his workplace, and move back and forth between the two services without having to log in each time. Or at home he might want to log in to an online travel agency, then book airline tickets online, and make hotel reservations online. It is a convenience for the user to be able to access all of these services without having to provide different user names and passwords at each service site. It is a valuable benefit to the user when he can do so safely, and knowing that his identity information is secure.

The Liberty Alliance Project was implemented to make this possible.

# The Liberty Alliance Project

In 2001 Sun Microsystems joined with other major companies to form the Liberty Alliance Project, the premier open standards organization for federated identity and identity-based services. The members of the Liberty Alliance Project represent some of the world's most recognized brand names and service providers. Liberty Alliance Project members drive products, services and partnerships across a spectrum of consumer and industrial products, financial services, travel, retailing, telecommunications and technology.

Access Manager implements two important sets of standards adopted by the Liberty Alliance Project: the Liberty Alliance Project frameworks, and the Security Assertions Markup Language (SAML) specifications. These implementations enable business partners to form a Circle of Trust.

## Liberty Alliance Frameworks

The Access Manager Federation Management feature is built upon Liberty Alliance frameworks. The Liberty Alliance Project developed the following specifications and guidelines for implementing complete network identity infrastructures and for deploying identity-based web services:

- Identity Federation Framework (ID-FF)
- Identity Web Services Framework (ID-WSF)
- Data Services Template (ID-WSF DST)
- Identity Services Interface Specifications (ID-SIS)

For more information these specifications, and listings of Liberty web service products, case studies, and white papers, see the Liberty Alliance Project website: `http://www.projectliberty.org/`

## The Circle of Trust

The goal of the Liberty Alliance Project is to enable individuals and organizations to easily conduct network transactions while protecting the individual's identity. This goal can be achieved only when commercial and non-commercial organizations join together into a *circle of trust*. In a circle of trust, service providers agree to join together in order to exchange user authentication information using Liberty web service technologies. This circle of trust must contain at least one identity provider, a service that maintains and manages identity information. The circle of trust also includes service providers that offer web-based services to users. Once a Circle Of Trust is established, single sign-on is enabled between all the providers.

In Access Manager, the circle of trust is known as an *authentication domain* although it is not a DNS domain. In Access Manger, an authentication domain describes entities that are grouped together for the purpose of identity federation.

A travel portal is a good example of an authentication domain. Typically, a travel portal is a website designed to help you find an access various travel service providers from one Internet location. The travel portal service forms a partnership with each hotel, airline, and car rental agency displayed on its website. The user logs into the travel portal and looks for a suitable hotel. When finished making hotel reservations, the user moves to the airline part of the travel portal to look for a suitable airline flight. This time, because of the partner agreement with the travel portal, the airline website shares the authentication information obtained earlier in the user's online session. The user moves from the hotel reservations website to the airline reservations website without having to re-authenticate. All of this is transparent to the user. The following figure illustrates the Circle of Trust formed among the travel portal, which acts as the Identity Provider, and each of the related business partners.

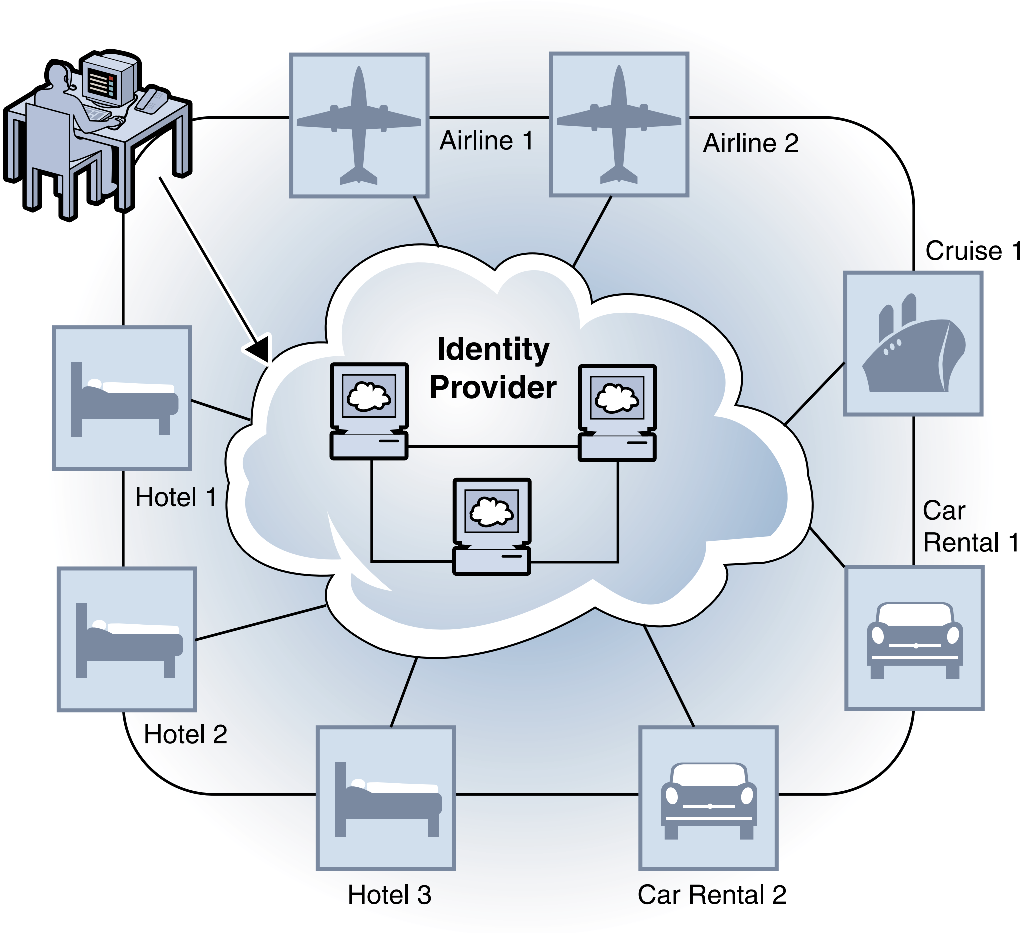


**FIGURE 5–1** The Circle of Trust

Account federation occurs when a user chooses to unite distinct service accounts and identity provider accounts. The user retains individual account information with each provider in the circle. At the same time, the user establishes a link that allows the exchange of authentication information between them. Users can choose to federate any or all identities they might have with the service providers that have joined this circle. When a user successfully authenticates with one service provider, she can access any of the her accounts within the circle of trust in a single session *without having to reauthenticate*.

# SAML Specifications

Access Manager uses the Security Assertion Markup Language (SAML) for exchanging security information. SAML resides within a system's security mechanisms to enable exchange of authentication and authorization information with other services. The SAML 1.0 specification set was submitted to the Organization for the Advancement of Structured Information Standards (OASIS) in March 2002 for standardization by the OASIS Security Services Technical Committee. OASIS is a not-for-profit, global consortium that drives the development, convergence and adoption of e-business standards.

SAML security information is expressed in the form of an assertion about a subject. A *subject* is an entity in a particular domain, either human or machine, with which the security information concerns itself. (A person identified by an email address is a subject as might be a printer.) An *assertion* is a package of verified security information that supplies one or more statements concerning a subject's authentication status, access authorization decisions or attributes. Assertions are issued by a SAML authority. (An *authority* is a platform or application that has been integrated with the SAML SDK, allowing it to relay security information.) The assertions are received by partner sites defined within the authority as *trusted*. SAML authorities use different sources to configure the assertion information including external data stores or assertions that have already been received and verified.

# Federation Management Implemented in Access Manager

In Access Manager, the Federation Management feature enables applications to participate in three different frameworks:

- Identity Federation Framework
- Identity Web Services Framework
- SAML 1.0 and 1.1 Specifications

These frameworks enable service providers to securely exchange authentication and authorization information. Client APIs are provided for web service consumers to communicate with web service providers. The following figure illustrates the internal architecture of a Liberty Web Services Consumer and a Web Service Provider.

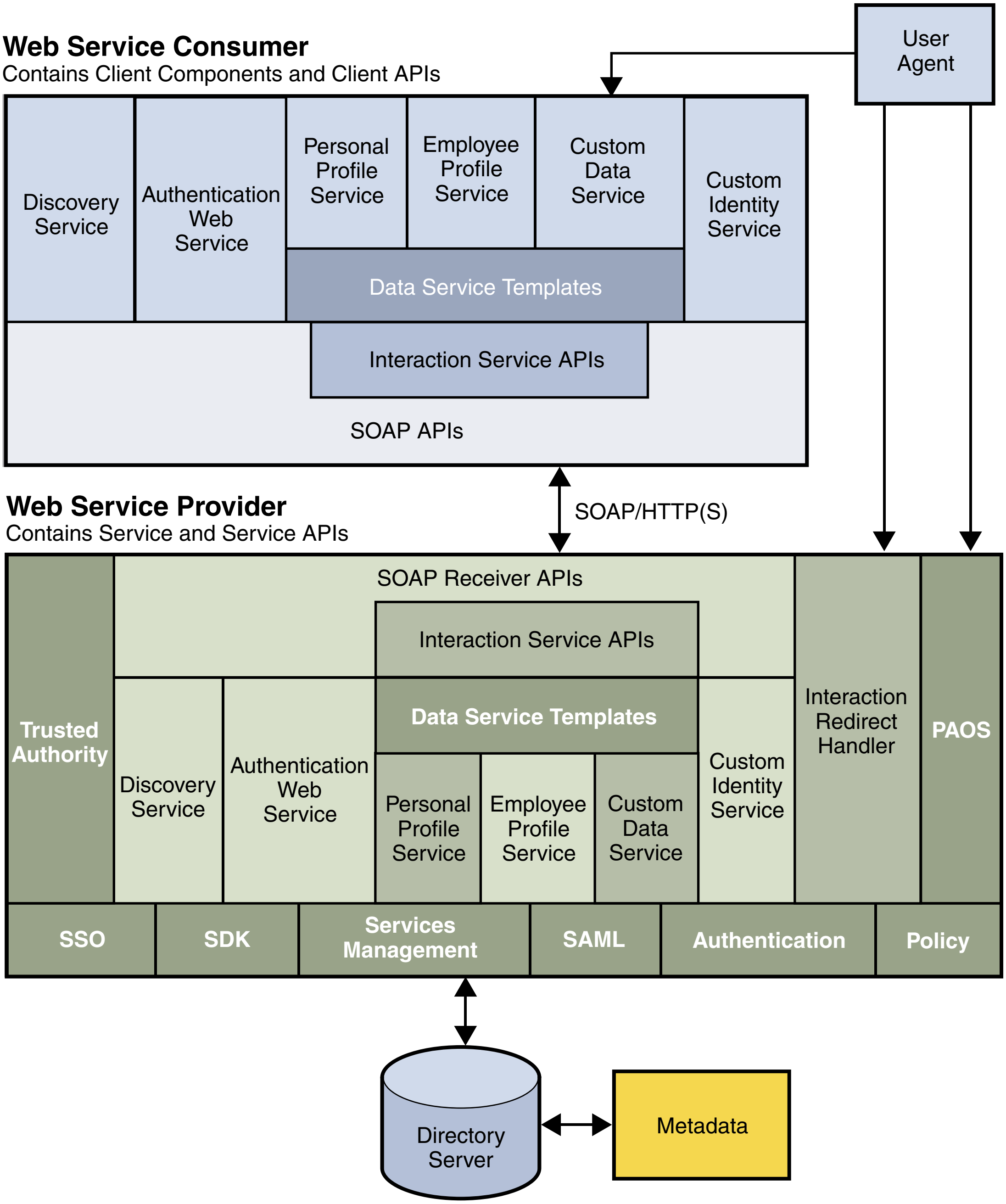


FIGURE 5–2 Web Services Consumer and Web Service Provider Architecture

The Web Service Consumer components and the Web Service Provider components were implemented in Access Manager 7.0. The components in the bottom layer of the Web Service Provider were implemented in Access Manager 6.1. These components include Single-Sign On (SS0), the Access Manager SDK, Service Management Services, SAML, Authentication modules, and a Policy Service. In the Identity Web Service Framework, the Data Service and Identity Service represent custom services that you can add to the Web Services Framework.

## Identity Federation Framework

The Identity Federation Framework (ID-FF) specifies core protocols, schema and concrete profiles that allow developers to create a standardized, multiple-vendor, identity federation network. These include the following:

Account linking termination.
: Users can choose to stop their account federation.

Authentication context.
: Service providers with federated accounts communicate the type and level of authentication that should be used when the user logs in.

Affiliation federation
: Federation based on group affiliation can be enabled in an authentication request. If enabled, it would indicate that the requester is acting as a member of the affiliation group identified. Federations are then established and resolved based on the affiliation, and not the requesting provider. The process allows for a unique identifier that represents the affiliation.

Dynamic identity provider proxying
: When one identity provider is asked to authenticate a principal that has already been authenticated by a second identity provider. In this case, the first identity provider may request authentication information from the second identity provider on behalf of the service provider. Proxy behavior can be controlled by indicating a list of preferred identity providers, and a value that defines the maximum number of proxy steps that can be taken. Proxy behavior is defined locally by the proxying identity provider, although a service provider controls whether or not to proxy.

Identity provider introduction.
: This feature provides the means for service providers to discover which identity providers a *principal* uses. A principal can be an organization or individual who interacts with the system. This is important when there are multiple identity providers in an identity federation network.

Name Identifier Mapping Protocol
: Defines how service providers can obtain name identifiers assigned to a principal that has federated in the name space of a different service provider. When a principal that has an identity federation relationship (and therefore a name identifier) with one service provider requests access to a second service provider site that requires a name identifier, the second service provider can use this protocol to obtain the identifier. It allows the requesting service provider to communicate with the second service provider about the principal even though no identity federation for the principal exists between them.

Name Registration
Enables a service provider or identity provider to register with each other a new name identifier for a principal at any time following federation.

One-time federation
The ability to federate for one session only can be enabled in an authentication request. This is useful for service providers with no user accounts, for principals who wish to act anonymously, or for dynamically-created user accounts. It allows for one-time federation, rather than a one-time name identifier for a session.

Opt-in account linking
Users can choose to federate different service provider accounts.

Single Sign-on and Federation Protocol
The protocol that defines the process that a user at a service provider goes through to authenticate their identity with an identity provider. It also specifies the means by which a service provider obtains an Authentication Assertion from an identity provider to allow single sign-on to the user. Two types of Single Sign-On exist which either the identity or service provider can implement:

- SOAP-based Single Sign On and Federation Protocol, which relies on a SOAP call from provider to provider. This is primarily the Browser Artifact SSO profile.
- Form POST-based Single Sign On and Federation Protocol, which rely on an HTTP form POST to communicate between providers.

Single Sign-Out Protocol
The protocol used to synchronize the session log-out functionality across all sessions that were authenticated and created by a particular identity provider. Two types of protocols exist which either the identity or service provider can implement:

- SOAP-based Single Log-Out Protocol relies on asynchronous SOAP messaging calls between providers.
- HTTP Redirect-based Single Log-Out Protocol

## Identity Web Services Framework

The Web Services Framework (ID-WSF) consists of a set of schema, protocols and profiles for providing a basic identity services, such as identity service discovery and invocation. Three parties are required for identity federation in a basic Liberty Web Services environment: a user agent, a web service consumer, and a web service provider.

The Web Services Framework consists of a set of schema, protocols and profiles for providing a basic identity services, such as identity service discovery and invocation. This framework includes the following:

Authentication Web Service
An identity service that enables a web service consumer to be authenticated using the Simple Authentication and Security Layer (SASL) mechanism. SASL defines a method for adding authentication support to connection-based protocols.

Discovery Service.
An identity service that allows a requester to discover resource offerings.

SOAP Binding.
A set of Java APIs for sending and receiving ID-* messages using SOAP and XML.

Security Mechanisms.
Defines a set of authentication mechanism and security properties which are factored into authorization decisions enforced by the targeting identity-based web services. Each mechanism contains both peer entity authentication (null/TLS/CClientTLS) and message authentication (null/X509/SAML).

Interaction Service.
A protocol for simple interaction of Web Services Framework participants with a Principal.

Trusted Authority.
APIs for creating security tokens used for authentication and authorization in Liberty II-enabled services.

Metadata Service.
A library of command-line tools for loading metadata into the Access Manager data store.

Reverse HTTP Bindings.
A protocol and set of APIs for retrieving data from Access Manager via clients such as cell phones.

## SAML Service

SAML defines an eXtensible Markup Language (XML) framework to achieve interoperability across different vendor platforms that provide SAML assertions. SAML is an XML framework for exchanging security information over the Internet. Access Manager SAML Service consists of a web service interface, a SAML core component, and a SAML framework that web services can connect to.

The Access Manager SAML Service enables the following functionality:

- Users can authenticate against Access Manager and access trusted partner sites without having to reauthenticate. This single sign-on process independent of the process enabled by Access Manager user session management.
- Access Manager acts 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.
- Access Manager acts as both an attribute authority (allowing trusted partner sites to query a subject's attributes) and an authentication authority (allowing trusted partner sites to query a subject's authentication information.)
- Two parties in different security domains can validate each other for the purpose of performing business transactions.
- Access Manager SAML APIs can be used to build Authentication, Authorization Decision and Attribute Assertions.

- The Access Manager SAML Service provides pluggable XML-based digital signature signing and verifying.

# Federation Management Protocols Flow

The following figure provides a high-level view of the system flow between various parties in a Liberty web services environment. A user agent, Service Provider, Identity Provider, and Personal Profile Service must be present in the environment. The figure and text illustrate the use of both Identity Federation Framework and Identity Federation Web Services Framework.

In this example:

- The web browser represents a user agent or a device used by an enterprise user.
- A Service Provider acts as a web services consumer (WSC) to invoke the web service on behalf of the user. The Service Provider relies on the Identity Provider authentication for single sign-on.
- The Identity Provider acts an authentication provider by authenticating the user and registering the user. The Identity Provider also acts a trusted authority, issuing security tokens through the Discovery Server.
- The Web Services provider serves requests from web services clients such as a Personal Profile Service provider.

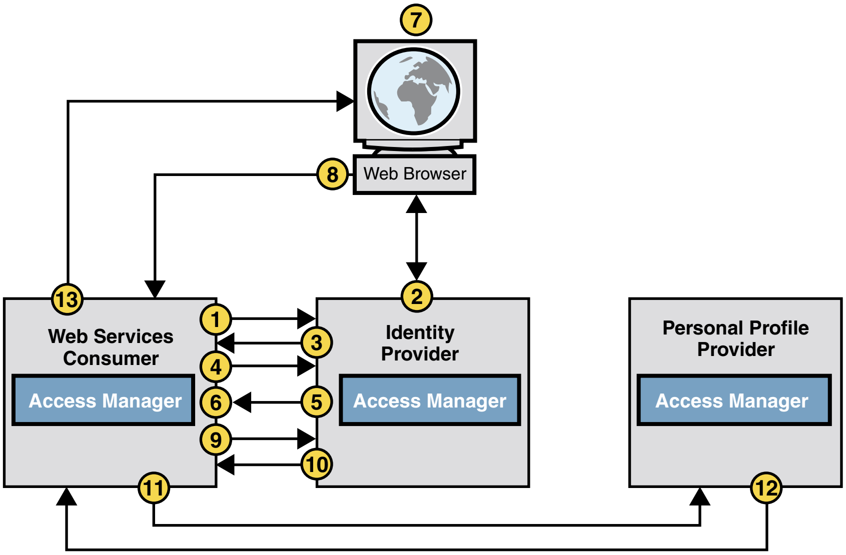


FIGURE 5–3 Identity Federation Protocols Flow

When a user logs into a circle of trust, the following events occur.

1. The Service Provider initiates the AuthnRequest.

   The request uses a browser artifact profile to contact the Single Sign-On service at the Identity Provider.

2. At the Identity Provider, the Single Sign-On service presents a login page to the user.

   The user enters credentials such as username and password.

3. Upon successful authentication, at the Identity Provider the Single Sign-On service sends an artifact to the Assertion Consumer service at the Service Provider.

4. The Identity Provider sends a SAML SOAP response to the Service Provider by keeping an authentication SML assertion in the response.

5. The Service Provider verifies the XML assertion and completes the Single Sign-On process.

   The assertion contains an attribute statement containing the Discover Service resource offering. The resource offering will be used as bootstrap information to invoke the Web Services Framework.

6. The user's browser, Service Provider and Identity Provider complete the Federation Single-Sign-On process.

   An assertion with an attribute statement containing the Discovery Service resource offering is included in the `ID-FF AuthnResponse`. This information can be used by any client to contact Discovery Service.

7. The user's browser requests access to services hosted on the Web Service Consumer.

   This requires contacting user's Personal Profile service.

8. The Web Service Consumer sends a discovery lookup query to the Discovery Service.

   The Web Service Consumer determines user's discovery resource offering from the bootstrap Assertion obtained earlier, then sends a discovery lookup query to the Discovery Service to determine where the user's Personal Profile instance is hosted.

9. The Discovery service returns a discovery lookup response to the Web Service Consumer.

   The lookup response contains the resource offering for the user's Personal Profile Service instance.

10. The Web Service Consumer sends a web services query that uses the protocol defined by the DataServiceTemplate. The web services query goes to the SOAP end point of the 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 Personal Profile Service resource offering must be followed.

11. The Personal Profile Service instance authenticates and validates authorization or policy, or both, for the requested user or Web Service Consumer, or for 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 Personal Profile Service instance returns a Data Services Template response to the Web Service Consumer after collecting all required data.

12. The Web Service Consumer processes the Personal Profile Service response, and then renders service pages containing the colleague's contact information to the user's browser.

For detailed information about all the components that are involved in Federation Management, see the *Sun Java System Access Manager 7.1 Federation and SAML Administration Guide*.

CHAPTER 6

# Access Manager Logging and Java Enterprise System Monitoring Framework

This chapter describes how Access Manager logging works, and provides some information about the Java Enterprise System (JES) Monitoring Framework. . The chapter contains the following sections:

- "How the Logging Feature Works" on page 81
- "Error and Access Logs" on page 85
- "Additional Logging Features" on page 86

Access Manager provides its own logging feature that records information such as user logins and logouts, session creation, policy evaluation, and other events. You can use Access Manager logs to audit system usage and to troubleshoot problems. Logging APIs enable external applications to access the Logging feature.

## How the Logging Feature Works

The Logging Service enables Access Manager services to record information such as access denials, access approvals, authentication events, and authorization violations. Administrators can use the logs to track user actions, analyze traffic patterns, and review authorization violations. The logged information from all Access Manager services are recorded in one centralized location. The default location for all Access Manager log files is `/var/opt/SUNWam/logs`.

### Logging Architecture

When Access Manager starts up or when any logging configuration data is changed through the console, logging configuration data is loaded into the Logging component. This data includes the log message format, log file name, maximum log size, and the number of history files. Applications can use the Client APIs to access the Logging features from a local or remote server. The Client APIs use an XML-over-HTTP layer to send logging requests to the Logging component on the server where Access Manager is installed.

### amLogging.xml

The Logging service stores the attributes and values for the logging function. A global service configuration file named `amLogging.xml` defines the Logging attributes. Examples of Logging Service attributes are maximum log size, log location, and log format (flat file or relational database). The attribute values are applied across the Access Manager deployment and inherited by every configured realm. By default, `amLogging.xml` is located in the directory `/etc/opt/SUNWam/config/xml`. The structure of `amLogging.xml` is defined by file `sms.dtd`.

## Log File Formats

Access Manager can record events in flat text files or in a relational database.

### Flat File Format

The default flat file format is the W3C Extended Log Format (ELF). Access Manager uses this format to record the default fields in each log record. See "Recorded Events" on page 84 for a list of default fields and their descriptions. The following code example illustrates an authentication log record formatted for a flat file.

**EXAMPLE 6–1** 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
```

In the example, the fields are in this order: `Time`, `Data`, `ModuleName`, `MessageID`, `Domain`, `ContextID`, `LogLevel`, `LoginID`, `IPAddr`, `LoggedBy`, and `HostName`.

### Relational Database Format

When Access Manager uses a relational database to log messages, the messages are stored in a database table. Access Manager uses Java Database Connectivity (JDBC) to access the database table. JDBC provides connectivity to a wide range of SQL databases. JDBC also provides access to other tabular data sources such as spreadsheets or flat files. Oracle® and MySQL databases are currently supported.

For log records generated by Access Manager, the `Data` and `MessageID` fields are used slightly differently than in previous Access manager versions. Starting with Access Manager, the `MessageID` field is introduced as a kind of template for types of log messages. For example, in previous versions, Access Manager would generate the following message in the `Data` field:

```
Data: "Created group
cn=agroupSubscription1,ou=Groups,dc=iplanet,dc=com"
```

In Access Manager, two log records are recorded for the one event:

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

and

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

The log records reflect the use of identities and realms. In this example, `CONSOLE-1` indicates an attempt to create an identity, and `CONSOLE-2` indicates the attempt to create an identity was successful. The root organization notation (`dc=iplanet,dc=com`) is replaced with 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 `MessagID` string is not internationalized in order to facilitate machine-readable analysis of the log records in any locale.

The following table summarizes the schema for a relational database.

**TABLE 6–1** Relational Database Log Format

| Column Name | Data Type | Description |
|---|---|---|
| `TIME` | VARCHAR(30) | Date of the log in the format `YYYY-MM-DD HH:MM:SS`. |
| `DATA` | VARCHAR(1024) | The variable data part of the log record pertaining to the MESSAGE ID. For MySQL, the Data Type is VARCHAR(255). |
| `MODULENAME` | VARCHAR(255) | Name of the Access Manager component invoking the log record. |
| `DOMAIN` | VARCHAR(255) | Access Manager 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. |
| `MESSAGE ID` | VARCHAR(255) | Non-internationalized message identifier for this log record's message. |
| `CONTEXT ID` | VARCHAR(255) | Identifier associated with a particular login session. |

## Log Files Directory

The log files record a number of events for each of the Access Manager components using the Logging Service. Administrators typically review these log files on a regular basis. The default location for all Access Manager log files is `/var/opt/SUNWam/logs`. The following table describes the files in the log files directory.

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`.

**TABLE 6–2** Files in the Log Files Directory

| File or Table | Information Logged |
|---|---|
| `amAuthLog` | Policy denies |
| `amPolicy.access` | Policy allows |
| `amPolicy.error` | Policy error events |
| `amConsole.access` | Successful console events |
| `amConsole.error` | Console error events |
| `amAuthentication.access` | Authentication successes |
| `amAuthentication.error` | authentication failures |
| `amPasswordReset.access` | Password reset events |
| `amSSO.access` | SSO creates/destroys |
| `amSAML.access` | SAML successful events |
| `amSAML.error` | SAML error events |
| `amLiberty.access` | Liberty successful events |
| `amLiberty.error` | Liberty error events |
| `amFederation.access` | Federation successful events |
| `amFederation.error` | Federation error events |
| `amAdmin.access` | amadmin CLI successful events |
| `amAdmin.error` | amadmin CLI error events |

## Recorded Events

The client passes the Logging Service logs information to the `com.sun.identity.log.LogRecord` class. The following table summarizes the items logged by default in the `LogRecord`.

TABLE 6–3 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. |
| Data | Variable data pertaining to the log records's MESSAGE ID. This field is not configurable. |
| Module Name | Name of the Access Manager service or application being logged. Additional information on the value of this field can be found in "Adding Log Data" on page 88. |
| Domain | Access Manager domain to which the user belongs. |
| Log Level | The Java 2 Platform, Standard Edition (J2SE) version 1.4 log level of the log record. |
| Login ID | ID of the user as the subject of the log record. The user ID is taken from the session token. |
| IP Address | IP address from which the operation was performed. |
| Logged By | User who writes the log record. The information is taken from the session token passed during `logger.log(logRecord, ssoToken).` |
| Host Name | Host name associated with the IP Address above. |
| MessageID | Non—internationalized message identifier for this log record's message. |
| ContextID | Identifier associated with a particular login session. |

# Error and Access Logs

Two types of Access Manager log files exist: access log files and error log files.

Access log files record general auditing information concerning the Access Manager deployment. A log may contain a single record for an event such as a successful authentication. A log may contain 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. Logging Service 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`.

For detailed reference information about events recorded in each type of Access Manager, log see the *Sun Java System Access Manager 7.1 Administration Guide*. The following table provides a brief description of the log file produced by each Access Manager component.

**TABLE 6–4** Access Manager Component Logs

| Component | Log Filename Prefix | Information Logged |
|---|---|---|
| Session | `amSSO` | Session management attributes values such as login time, logout time, timeout limits. |
| Administration Console | `amConsole` | User actions performed through the administration console such as creation, deletion and modification of identity-related objects, realms, and policies. |
| Authentication | `amAuthentication` | User logins and logouts. |
| Identity Federation | `amFederation` | Federation-related events such as the creation of an Authentication Domain and the creation of a Hosted Provider. The federation logs are prefixed with `amFederation`. |
| Authorization (Policy) | `amPolicy` | Policy-related events such as policy creation, deletion, or modification, and policy evaluation. |
| Policy Agent | `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 Access Manager machine in the Authentication logs. |
| SAML | `amSAML` | SAML-related events such as assertion and artifact creation or removal, response and request details, and SOAP errors. |
| Command-line | `amAdmin` | Event errors that occur during operations using the command line tools. Examples are: loading a service schema, creating policy, and deleting users. |

# Additional Logging Features

You can enable a number of logging features for added functionality. The additional features include secure logging, command-line logging, and remote logging.

## Secure Logging

This feature adds an extra measure of security to the logging feature. When secure logging is enabled, the Logging component 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 Access Manager. 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 the *Sun Java System Access Manager 7.1 Administration Guide*.

## Remote Logging

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

- When the login URL in the Naming Service of an Access Manager instance points to a remote Access Manager instance, and a trust relationship between the two instances has been configured.
- When the Access Manager APIs are installed in a remote Access Manager instance, and a client application or a simple Java class running on the Access Manager server uses the logging APIs.
- When logging APIs are used by Access Manager agents.

## Log Reading

Access Manager provides Logging APIs for writing your own custom log reading program. You can set up queries to retrieve specific records from the log file or database. This is useful for auditing purposes. For more information, see the *Sun Java System Access Manager 7.1 Developer's Guide*.

# Java Enterprise System Monitoring Framework

Access Manager 7.1 integrates with the Java Enterprise System (JES) monitoring framework through Java Management Extensions (JMX). JMX technology provides the tools for building distributed, Web-based, modular, and dynamic solutions for managing and monitoring devices, applications, and service-driven networks. Typical uses of the JMX technology include: consulting and changing application configuration, accumulating statistics about application behavior, notification of state changes and erroneous behaviors. Data is delivered to centralized monitoring console. Access Manager 7.1 uses the Java ES Monitoring Framework to capture statistics and service-related data such as the following:

- Number of attempted, successful, and failed authentications
- Number of active sessions, statistics from session failover DB
- Session failover database statistics
- Policy caching statistics
- Policy evaluation transaction times
- Number of assertions for a given provider in a SAML/Federation deployment Web Service Security

For comprehensive information about how the JES monitoring framework works and how you can use the monitoring framework with Access Manager, see the *Sun Java Enterprise System 2006Q4 Monitoring Guide*.

# Index