

For Review Purposes Only

Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for Web Agents



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

Part No: 820-5816-10
October 31, 2008

Copyright 2008 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more U.S. patents or pending patent applications in the U.S. and in other countries.

U.S. Government Rights – Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

This distribution may include materials developed by third parties.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, the Solaris logo, the Java Coffee Cup logo, docs.sun.com, Java, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. or its subsidiaries in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and SunTM Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Products covered by and information contained in this publication are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical or biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2008 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. Tous droits réservés.

Sun Microsystems, Inc. détient les droits de propriété intellectuelle relatifs à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et ce sans limitation, ces droits de propriété intellectuelle peuvent inclure un ou plusieurs brevets américains ou des applications de brevet en attente aux Etats-Unis et dans d'autres pays.

Cette distribution peut comprendre des composants développés par des tierces personnes.

Certains composants de ce produit peuvent être dérivées du logiciel Berkeley BSD, licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays; elle est licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, le logo Solaris, le logo Java Coffee Cup, docs.sun.com, Java et Solaris sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc., ou ses filiales, aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui, en outre, se conforment aux licences écrites de Sun.

Les produits qui font l'objet de cette publication et les informations qu'il contient sont régis par la législation américaine en matière de contrôle des exportations et peuvent être soumis au droit d'autres pays dans le domaine des exportations et importations. Les utilisations finales, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes chimiques ou biologiques ou pour le nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou réexportations vers des pays sous embargo des Etats-Unis, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exclusive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une façon directe ou indirecte, aux exportations des produits ou des services qui sont régis par la législation américaine en matière de contrôle des exportations et la liste de ressortissants spécifiquement désignés, sont rigoureusement interdites.

LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFACON.

List of Remarks

REMARK 2-3	Reviewer	I'm unclear about the reach of the <code>ssoadm</code> command-line utility. Can it be used to edit properties in the bootstrap file? 36
REMARK 2-4	Reviewer	I suppose that I should mention policy response attributes here, no? 37
REMARK 3-1	Reviewer	Is the above description accurate? 53
REMARK 3-2	Reviewer	What's with this property? It's not in the Console. Is it still valid? 55
REMARK 3-3	Reviewer	What's with this property? It isn't available from the Console. 59
REMARK 3-4	Reviewer	What's with this property? It isn't available from the Console. 60
REMARK 4-1	Reviewer	Above, I say one can set properties in the bootstrap file using the <code>ssoadm</code> utility. Is that true? 61
REMARK 4-2	Reviewer	Does the explanation above make sense? 71
REMARK 4-3	Reviewer	Is the sentence above correct? Also, I don't know what the note below is trying to say. It talks about a value being fetched, but I don't know which value. Anyway, part of the explanation is missing here. 73
REMARK 4-4	Reviewer	Is the preceding property accurate? It's in the Deprecated Agent Properties section. 74
REMARK 4-5	Reviewer	The term used before for the agent profile password was "shared secret." I'm assuming that term is obsolete at this point. Am I correct? Should I be calling this the "agent profile password"? 75
REMARK 4-6	Reviewer	I'm not sure if all the stuff in this section still applies. For example, do the scripts such as <code>crypt_util</code> still apply? What about the steps in general? 75
REMARK 4-7	Reviewer	It seems this might have changed in 3.0. What's missing and/or wrong from this explanation? 77
REMARK 4-8	Reviewer	Is it true that this property is enabled by default? On my deployment, it is not enabled. Did this change for 3.0? 77
REMARK 4-9	Reviewer	Is this default correct? 78
REMARK 4-10	Reviewer	Below are sections on Composite advice and Malicious headers. These sections were in 2.2 as new features. They seem useful for 3.0, but I'm not certain how to incorporate them into this guide. Right now, no tasks are related to these features. I'm not sure if there should be another section, such as "More Features of Web Agents in the Policy Agent 3.0 Software Set." Or maybe these could be incorporated into FAQs. I'm not sure. 80
REMARK B-1	Reviewer	Is this write up accurate? Does everything in this wildcard section fit the agent type? .. 85

REMARK D-1	Reviewer	Please look at the command above. Are all the details of it accurate.It seems that the above example is different for web agents and J2EE agents. Besides that example above, does this appendix apply equally to web agents and J2EE agents? Are there any differences that should be addressed? 101
------------	----------	---

Contents

Preface	9
1 Introduction to Policy Agent 3.0	15
Overview of New Features in Policy Agent 3.0	15
Compatibility and Coexistence of Policy Agent 3.0 with Previous Releases	17
Compatibility of Policy Agent 3.0 with Access Manager 7.1 and Access Manager 7 2005Q4	17
Coexistence of Policy Agent 3.0 With Policy Agent 2.2	17
2 Role of Policy Agent 3.0 Software	19
An Overview of Policy Agent 3.0	19
A Generalized Example of the Policy Decision Process	23
Examples of the Policy Decision Process by Agent Type	28
Policy Agent 3.0 and the Centralized Agent Configuration Option	33
Web Agents and J2EE Agents: Similarities and Differences	34
Focusing on Web Agents in the Policy Agent 3.0 Release	39
Uses of Web Agents	39
How Web Agents Work	39
3 Vital Installation Information for a Web Agent in Policy Agent 3.0	41
Format of the Distribution Files for Web Agents in Policy Agent 3.0	41
Introduction of the agentadmin Program in Web Agents for Policy Agent 3.0	41
agentadmin --install	42
agentadmin --uninstall	44
agentadmin --listAgents	45
agentadmin --agentInfo	46
agentadmin --version	47

agentadmin --uninstallAll	47
agentadmin --usage	48
agentadmin --help	49
Web Agent Directory Structure in Policy Agent 3.0	50
Location of the Web Agent Base Directory in Policy Agent 3.0	50
Inside the Web Agent Base Directory in Policy Agent 3.0	50
Policy Agent 3.0: Web Agent Properties	53
Web Agent Properties in the OpenSSOAgentBootstrap.properties File	54
Web Agent Properties Available Using the OpenSSO Enterprise Console or Other Methods	54
4 Common Web Agent Tasks and Features in Policy Agent 3.0	61
Providing Failover Protection for a Web Agent	63
Changing the Web Agent Caching Behavior	63
Cache Updates	64
Hybrid Cache Updates	64
Configuring the Not-Enforced URL List	65
Configuring the Not-Enforced IP Address List	66
Enforcing Authentication Only	66
Providing Personalization Capabilities	66
Providing Personalization With Session Attributes	67
Providing Personalization With Policy-Based Response Attributes	68
Providing Personalization With User Profile Attributes Globally	69
Setting the Fully Qualified Domain Name	70
Turning Off FQDN Mapping	71
Resetting Cookies	72
Configuring CDSSO	72
Setting the REMOTE_USER Server Variable	73
Setting Anonymous User	74
Validating Client IP Addresses	74
Resetting the Agent Profile Password	74
▼ To Reset the Agent Profile Password on Solaris Systems	75
▼ To Reset the Agent Profile Password on Windows Systems	76
▼ To Reset the Agent Profile Password on Linux Systems	76
Configuring Web Agent Log Rotation	77

Enabling Load Balancing	78
Load Balancer in Front of OpenSSO Enterprise	78
Load Balancer in Front of the Web Agent	79
Load Balancers in Front of Both the Web Agent and OpenSSO Enterprise	80
Composite Advice	80
Malicious Header Attributes Automatically Cleared by Agents	80
A Silent Installation and Uninstallation of a Web Agent in Policy Agent 3.0	81
About Silent Installation and Uninstallation of a Web Agent in Policy Agent 3.0	81
Generating a State File for a Web Agent Installation	81
Using a State File for a Web Agent Silent Installation	82
Generating a State File for a Web Agent Uninstallation	83
Using a State File for a Web Agent Silent Uninstallation	84
B Wildcard Matching in Policy Agent 3.0 Web Agents	85
The Multi—Level Wildcard: *	86
The One-Level Wildcard: -*	87
C Precautions Against Session-Cookie Hijacking in an OpenSSO Enterprise Deployment	89
Defining Key Cookie Hijacking Security Issues	89
OpenSSO Enterprise Solution: Cookie Hijacking Security Issues	93
OpenSSO Enterprise Solution: Shared Session Cookies	93
OpenSSO Enterprise Solution: A Less Secure Application	94
OpenSSO Enterprise Solution: Modification of Profile Attributes	94
Key Aspects of the OpenSSO Enterprise Solution: Cookie Hijacking Security Issues	94
Implementing the OpenSSO Enterprise Solution for Cookie Hijacking Security Issues	96
Updating an Agent Profile	96
Configuring the OpenSSO Enterprise Deployment Against Cookie Hijacking	97
D Using the ssoadm Command-Line Utility With Agents	101
An ssoadm Command-Line Example Specific to Agents	101
Listing the Options for an ssoadm Subcommand	102
Agent-Related Subcommands for the ssoadm Command	104
The ssoadm Command: add-agent-to-grp subcommand	104

The ssoadm Command: agent-remove-props subcommand	105
The ssoadm Command: create-agent subcommand	106
The ssoadm Command: create-agent-grp subcommand	107
The ssoadm Command: delete-agent-grps subcommand	108
The ssoadm Command: delete-agents subcommand	109
The ssoadm Command: list-agent-grp-members subcommand	110
The ssoadm Command: list-agent-grps subcommand	111
The ssoadm Command: list-agents subcommand	111
The ssoadm Command: remove-agent-from-grp subcommand	112
The ssoadm Command: show-agent subcommand	113
The ssoadm Command: show-agent-grp subcommand	114
The ssoadm Command: show-agent-membership subcommand	115
The ssoadm Command: show-agent-types subcommand	116
The ssoadm Command: update-agent subcommand	116
The ssoadm Command: update-agent-grp subcommand	117
 E Web Agent Error Codes	119
Error Code List	119
 F Developing Your Own OpenSSO Enterprise Web Agent	123
The Web Agent Development Toolkit	123
 Index	125

Preface

The Sun OpenSSO Enterprise Policy Agent software consists of J2EE (Java 2 Platform Enterprise Edition) agents and web agents. This *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for Web Agents* provides an overview of how web agents work in the Sun OpenSSO Enterprise Policy Agent 3.0 release, while also providing a description of the features and processes of Policy Agent that are the same for J2EE agents and web agents in general. J2EE agents and web agents have many similarities, but the two types of agents also have some differences. This guide starts with a summary of the similarities and differences. Then this guide follows with information only applicable to web agents. However, for information for specific web agents, such as installation information and agent-specific configuration, see the individual web agent guide for that agent.

Within the Policy Agent documentation set, each agent has its own guide. Therefore, each book specific to a web agent covers aspects that are unique to that particular web agent.

Who Should Use This Book

This *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for Web Agents* is intended for use by IT professionals who manage access to their network. Administrators should understand the following technologies:

- Directory technologies
- JavaServer Pages™ (JSP) technology
- HyperText Transfer Protocol (HTTP)
- HyperText Markup Language (HTML)
- eXtensible Markup Language (XML)

Before You Read This Book

You should be familiar with documentation related to Policy Agent 3.0. Sun Microsystems server documentation sets, some of which are mentioned in this preface, are available at <http://docs.sun.com>:

OpenSSO Enterprise Documentation Set

Policy Agent 3.0 is being introduced with OpenSSO Enterprise 8.0, documents in the OpenSSO Enterprise 8.0 documentation set are available at the following location:

Be aware that in the future, newer versions of OpenSSO Enterprise might also apply to Policy Agent 3.0. In such cases, see the appropriate OpenSSO Enterprise documentation set.

- OpenSSO Enterprise 8.0 Installation Instructions
- OpenSSO Enterprise 8.0 Core Documentation

Updates to the *Release Notes* and links to modifications of the core documentation can be found in the OpenSSO Enterprise documentation collection.

Policy Agent 3.0 Documentation Set

This *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for Web Agents* is available in two documentation sets: the OpenSSO Enterprise documentation set and in the Policy Agent 3.0 documentation set as described in this section. The other guides in the Policy Agent 3.0 documentation set are described in the following sections:

- [“Individual Agent Guides” on page 10](#) (each agent has its own guide)
- [“Release Notes” on page 11](#)

Individual Agent Guides

The individual agents in the Policy Agent 3.0 software set are available on a different schedule than OpenSSO Enterprise itself. Therefore, documentation for OpenSSO Enterprise and Policy Agent are available in separate sets, except for the two user's guides, which are available in both documentation sets.

The documentation for the individual agents is divided into two subsets: a web agent subset and a J2EE agent subset.

Each web agent guide provides agent-specific information about a particular web agent, such as installation and configuration information.

Each J2EE agent guide provides agent-specific information about a particular J2EE agent, such as installation and configuration information.

The individual agent guides are listed along with supported server information in the Policy Agent 3.0 Release Notes.

Release Notes

The Policy Agent 3.0 Release Notes are released online after an agent or set of agents is released. The release notes include a description of what is new in the current release, known problems and limitations, installation notes, and how to report issues with the software or the documentation.

Related Sun Microsystems Product Documentation

The information in the table that follows specifies the key document collections that might be involved in anOpenSSO Enterprise deployment, which are available at the following location:

<http://docs.sun.com/prod/entsys.05q4>

TABLE P-1 Documentation Collections Related to

Title	Location
Directory Server:	http://docs.sun.com/coll/1316.1
Web Server:	http://docs.sun.com/coll/1308.1
Application Server:	http://docs.sun.com/coll/1310.1
Message Queue:	http://docs.sun.com/coll/1307.1
Web Proxy Server:	http://docs.sun.com/coll/1311.1

Accessing Sun Resources Online

For product downloads, professional services, patches and support, and additional developer information, go to the following:

Download Center

<http://www.sun.com/software/download>

Sun Enterprise Services, Solaris Patches, and Support

<http://sunsolve.sun.com/>

Developer Information

<http://developers.sun.com/prodtech/index.html>

Contacting Sun Technical Support

If you have technical questions about this product that are not answered in the product documentation, go to:

<http://www.sun.com/service/contacting>

Related Third-Party Web Site References

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

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions.

To share your comments, go to (<http://docs.sun.com>) and click Send Comments. In the online form, provide the 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 guide or at the top of the document.

For example, the title of this guide is *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for Web Agents*, and the part number is 820-5816.

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-2 Typographic Conventions

Typeface	Meaning	Example
AaBbCc123	The names of commands, files, and directories, and onscreen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>machine_name%</code> you have mail.
AaBbCc123	What you type, contrasted with onscreen computer output	<code>machine_name%</code> su Password:
<i>aabbcc123</i>	Placeholder: replace with a real name or value	The command to remove a file is <i>rm filename</i> .
<i>AaBbCc123</i>	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the <i>User's Guide</i> . A <i>cache</i> is a copy that is stored locally. Do <i>not</i> save the file. 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-3 Shell Prompts

Shell	Prompt
C shell	<code>machine_name%</code>
C shell for superuser	<code>machine_name#</code>
Bourne shell and Korn shell	<code>\$</code>
Bourne shell and Korn shell for superuser	<code>#</code>

Introduction to Policy Agent 3.0

This chapter introduces Policy Agent 3.0. The 8.0 release of OpenSSO Enterprise server and the 3.0 release of Policy Agent software were developed simultaneously and, therefore, are closely integrated. In fact, the Policy Agent 3.0 software set is more closely connected to the server (OpenSSO Enterprise) than ever before, making for a simplified administrative experience.

The sections that follow in this chapter highlight what is new in Policy Agent for the 3.0 release while also discussing the topic of compatibility as related to Policy Agent 3.0.

Overview of New Features in Policy Agent 3.0

Policy Agent 3.0 has the following new features and improvements:

- Centralized agent configuration

The centralized agent configuration feature moves most of the agent configuration properties from the `AMAgent.properties` file to the OpenSSO Enterprise central data repository. An agent administrator can then manage the multiple agent configurations from a central server location, using either the OpenSSO Enterprise Administration Console or the `ssoadm` command-line utility. The agent administrator no longer needs to edit an agent's `AMAgent.properties` file.

The centralized agent configuration feature separates the Policy Agent 3.0 configuration data into two sets:

- The properties required for the agent to start up and initialize itself are stored in the `OpenSSOAgentBootstrap.properties` file locally on the server where the agent is installed. For example, the agent profile name and password used to access the OpenSSO Enterprise server are stored in the bootstrap file.
 - The rest of the agent properties are stored either centrally in the OpenSSO Enterprise data repository (centralized configuration option) or locally in the `OpenSSOAgentConfiguration.properties` file (local configuration option).
- Agent groups

You can assign agents of the same type (J2EEAgent or WebAgent) from the Policy Agent 3.0 software set to an agent group. All agents in a group then selectively share a common set of configuration properties. Thus, the agent configuration and management are simplified because an administrator can manage all of the agents within a group as a single entity.

Although all agents in the same group can share the same properties, defining a few specific properties (for example, the notification URL or agent URI properties) for individual agents is probably necessary.

- More hot-swappable agent configuration properties

Version 3.0 agents have more hot-swappable configuration properties. An administrator can change a hot-swappable configuration property value for an agent without having to restart the agent's deployment container for the new value to take effect. Properties in the `OpenSSOAgentBootstrap.properties` file are not hot-swappable.

- One-level wildcard support in URL policy

While the regular wildcard support applies to multiple levels in a resource, the one-level wildcard applies to only the level where it appears in a resource. For more information, see [Appendix B, “Wildcard Matching in Policy Agent 3.0 Web Agents”](#)

- Default J2EE agent installation option with minimal questions asked during the installation

Default or custom installation:

- **Default** (`agentadmin -install`): The `agentadmin` program displays a minimal number of prompts and uses default values for the other options. Use the default install option when the default option meets your deployment requirements.
- **Custom** (`agentadmin -custom-install`): The `agentadmin` program displays a full set of prompts, similar to those presented by the Policy Agent 2.2 installer. Use the custom install option when you want to specify values other than the default options.

- Option to create the agent profile for J2EE agents in the server during installation

The Policy Agent 3.0 installer supports an option to create the agent profile in the OpenSSO Enterprise server during the agent installation so you don't have to create the profile manually using the OpenSSO Enterprise Console or the `ssoadm` utility.

- Option to lock the agent configuration properties

Changing configuration properties can have unexpected results. Furthermore, hot-swappable properties take effect immediately. Therefore, configuration mistakes are instantly implemented. In the Policy Agent 3.0 release you have a method for locking the configuration to help prevent such accidental changes.

The Policy Agent 3.0 properties

- Automated migration support

You can migrate a version 2.2 agent to a version 3.0 agent using the `agentadmin` program with the `-migrate` option.

Note: OpenSSO Enterprise does not support version 2.1 policy agents.

Compatibility and Coexistence of Policy Agent 3.0 with Previous Releases

This section consists of information about the compatibility and coexistence of the web agents in the Policy Agent 3.0 software set with previous releases of both Access Manager and Policy Agent.

Web Agents in the Policy Agent 3.0 release are compatible with versions of Access Manager as described in this section.

Compatibility of Policy Agent 3.0 with Access Manager 7.1 and Access Manager 7 2005Q4

Access Manager 7.1 and Access Manager 7 2005Q4 are compatible with Policy Agent 3.0. However, because Access Manager does not support centralized agent configuration, an agent in the 3.0 release deployed with Access Manager must store its configuration data locally in the `OpenSSOAgentConfiguration.properties` files.

The `com.sun.am.policy.agents.config.repository.location` property in the OpenSSO Enterprise server Agent Service schema (`AgentService.xml` file) specifies where the agent configuration data is stored:

- `local`: Configuration data is stored locally in the `OpenSSOAgentConfiguration.properties` files on the server where the agent is deployed.
- `centralized`: Configuration data is stored in the OpenSSO Enterprise centralized data repository.

For both configurations, the `OpenSSOAgentBootstrap.properties` file on the server where the agent is deployed contains the information required for the agent to start and initialize itself.

Coexistence of Policy Agent 3.0 With Policy Agent 2.2

OpenSSO Enterprise supports both Policy Agent 3.0 and Policy Agent 2.2 in the same deployment. However, agents in the 2.2 release must continue to store their configuration data locally in the `AMAgent.properties` file. And because the configuration data of agents in the 2.2 release is local to the agent, OpenSSO Enterprise centralized agent configuration is not supported. To configure an agent in the Policy Agent 2.2 release, you must continue to edit the `AMAgent.properties` file.

For more information about Policy Agent 2.2, see the documentation collection:
<http://docs.sun.com/coll/1322.1>

Role of Policy Agent 3.0 Software

OpenSSO EnterprisePolicy Agent 3.0 software consists of web agents and J2EE agents. This chapter explains the similarities and differences of these two types of agents. Then the chapter describes web agents in more detail.

An Overview of Policy Agent 3.0

Access control in Sun OpenSSO Enterprise is enforced using agents. Agents protect content on designated deployment containers, such as web servers and application servers, from unauthorized intrusions. Agents are separate from OpenSSO Enterprise.

Note – The most current agents in the Policy Agent software set can be downloaded from the Identity Management page of the Sun Microsystems Download Center:

<http://www.sun.com/software/download>

Web agents and J2EE agents differ in a few ways. One significant way the two agent types differ is in the resources that the two agent types protect. Web agents protect resources on web and proxy servers while J2EE agents protect resources on application and portal servers. However, the most basic tasks that the two agent types perform in order to protect resources are similar.

This chapter does the following:

- Explains what agents do.

- Describes briefly what a web agent is.

- Describes briefly what a J2EE agent is.

- Explains how these two types of agents are similar to each other and yet different.

All agents do the following:

- Enable cross-domain single sign-on (CDSSO).
- Determine whether a user is authenticated.
- Determine whether a resource is protected.
- For an authenticated user attempting to access a protected resource, determine whether the user is authorized to access that resource.
- Allow or deny a user access to a protected resource according to the results of the authentication and authorization processes.
- Log access information and diagnostic information.

The preceding task descriptions provide a simplified explanation of what agents do. Agents perform these tasks in conjunction with OpenSSO Enterprise. More specifically, agents work with various OpenSSO Enterprise services, such as Authentication Service, Session Service, Logging Service, and Policy Service to perform these tasks. Both agent types, J2EE agents and web agents interact with these OpenSSO Enterprise services in a similar manner as depicted by the following figures.

The figure that follows is J2EE-agent specific.

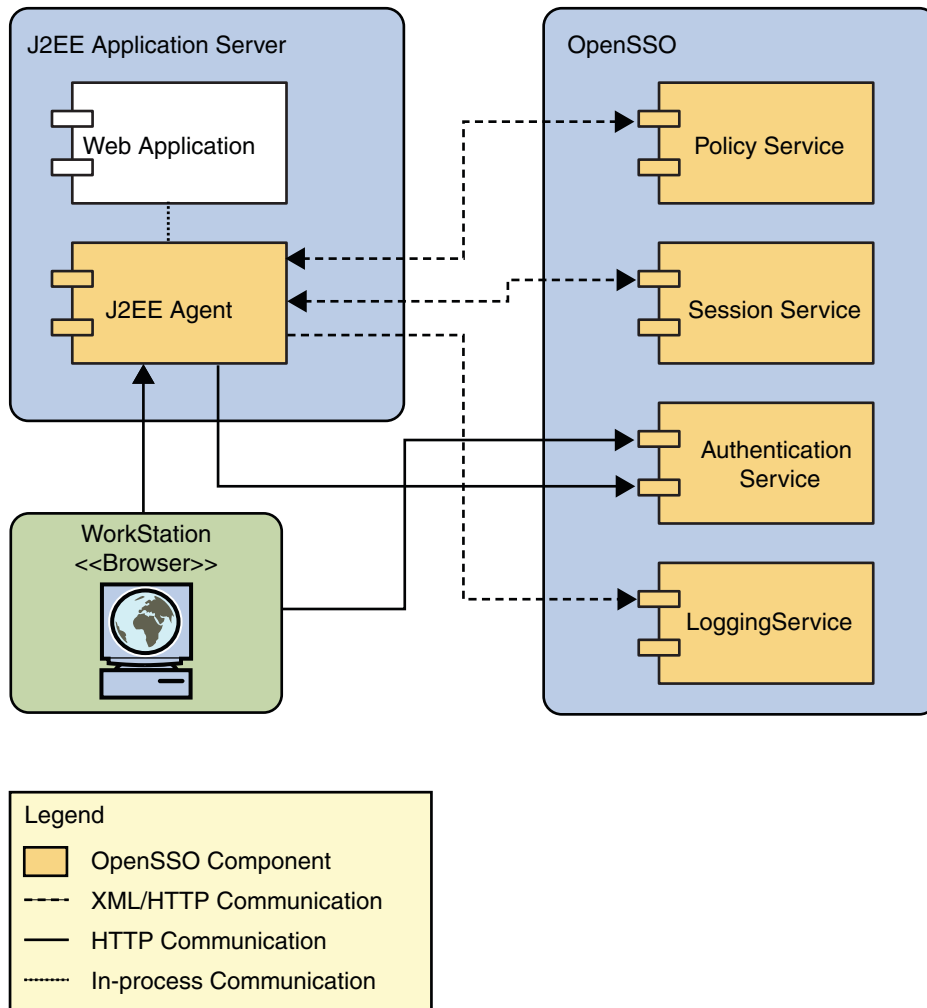


FIGURE 2-1 J2EE Agent Interaction with OpenSSO Enterprise Services

Notice the slight differences in how each agent type (J2EE agents in the preceding figure and web agents in the following figure) interacts with the OpenSSO Enterprise Services. Therefore, the two agent types tend to achieve the same results, but sometimes with slightly different processes.

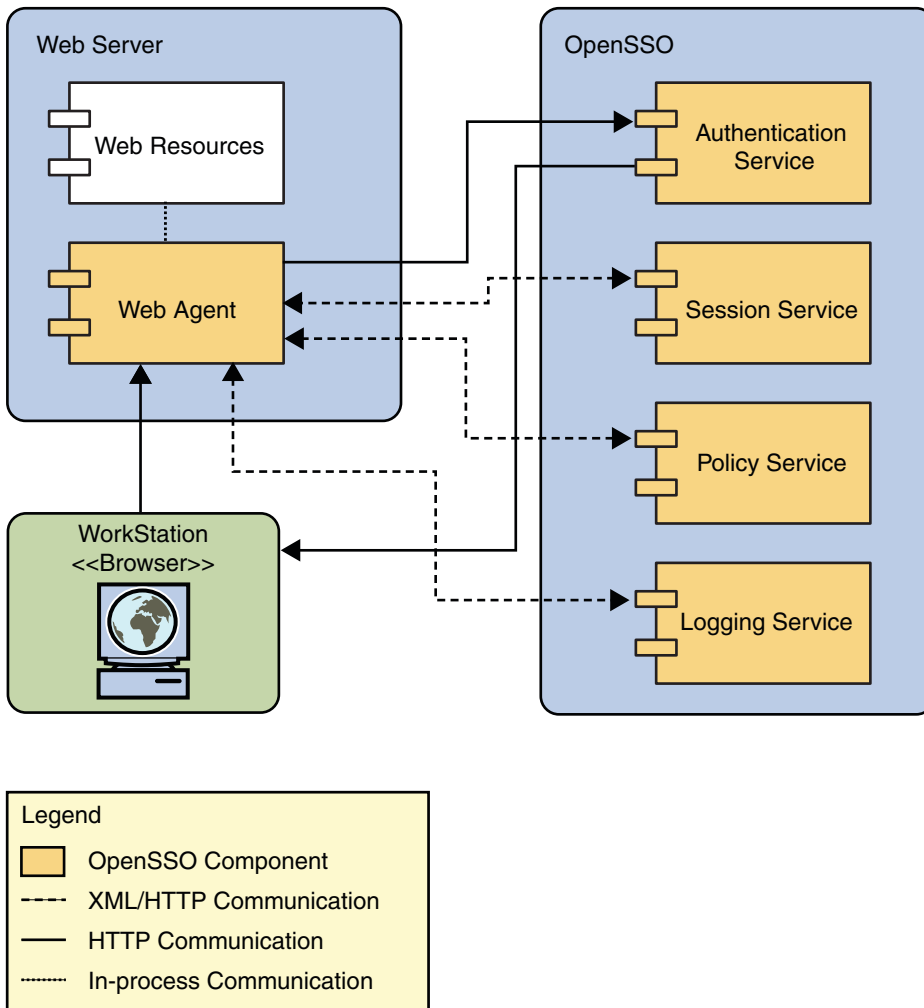


FIGURE 2-2 Web Agent Interaction with OpenSSO Enterprise Services

A key point is that the agent must interact continuously with various OpenSSO Enterprise services. For both J2EE agents and web agents, the interaction starts between the agent and the OpenSSO Enterprise Authentication Service. After authentication, users still cannot access a protected resource until the defined policies regarding user privileges are approved. The agent and OpenSSO Enterprise continue to interact, performing several small tasks back and forth, until the agent finally enforces a policy decision to either allow or deny access. The interactions that take place between Policy Agent and OpenSSO Enterprise are not covered in detail in Policy Agent documentation. For a more information on such interactions, see [Sun OpenSSO Enterprise 8.0 Administration Guide](#).

A Generalized Example of the Policy Decision Process

When a user attempts to access content on a protected resource, many deployment variables are involved. For example, firewalls might be present or load balancers might be involved. From a high level, the two agent types (J2EE agents and web agents) would seem to handle these deployments in the same manner. For example, observe how the two figures that follow are nearly identical. In these two figures J2EE agents and web agents incorporate firewalls and load balancers in the same manner.

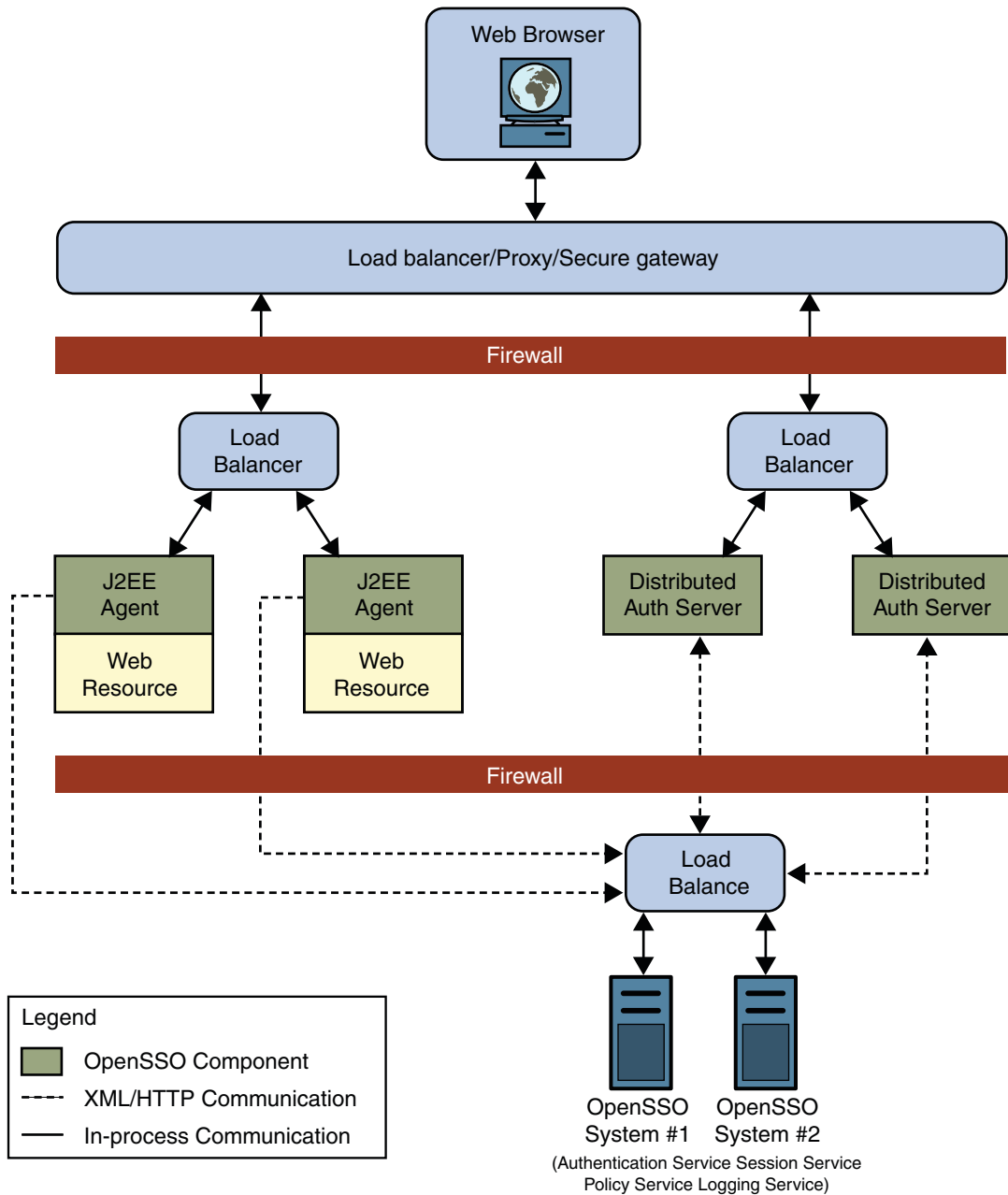


FIGURE 2-3 High Level View of an OpenSSO Enterprise Deployment With J2EE Agents, Firewalls, and Load Balancers

By comparing the preceding figure with the figure that follows, you can see that from this level of detail J2EE agents and web agents do not differ in the general role they play in an OpenSSO Enterprise deployment. At the policy decision level, the differences between the two agent types are more easily observed, and components such as firewalls and load balancers can add complexity to the policy decision process. Therefore, for the basic example provided in this section about the policy decision process, such complex details are not included.

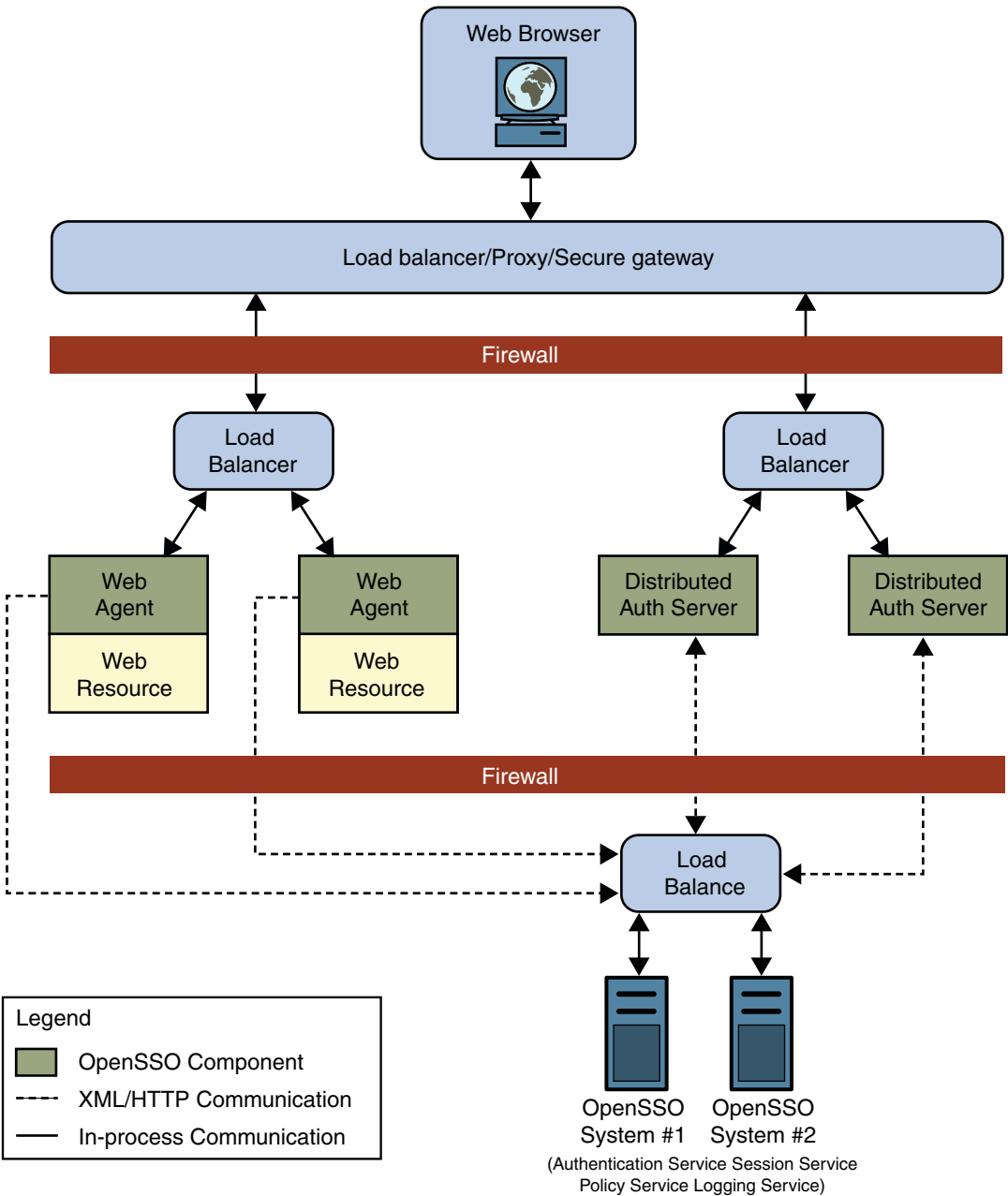


FIGURE 2-4 High Level View of an OpenSSO Enterprise Deployment With Web Agents, Firewalls, and Load Balancers

Another example of a deployment variable concerns authentication levels. In a real-world deployment, different resources on a deployment container (such as an application or web server) might require different levels of authentication. Suffice to say a great deal of complexity is involved in providing a generic example of a policy decision process, especially one that applies equally to J2EE agents and web agents. For one, the process varies greatly depending on the specifics of the deployment. Many other factors can affect the policy decision process, such as the IP address, time zone, and policy expiration time.

Each deployment variable can add a layer of complexity, which might affect how an agent reacts and how OpenSSO Enterprise reacts. This section provides a simple example of a policy decision process that highlights the role of an agent. Therefore, many of the detailed tasks and interactions, especially those processes that occur in OpenSSO Enterprise are left out. Do not expect the deployment represented in this example to match the deployment at your site. This is a generalized example that is applicable to both web and J2EE agents. Some of the basic steps in the policy decision process are depicted in [Figure 2-5](#). The figure is followed by a written description of the process. Notice that the figure illustrates the policy decision process in terms of the components the decision passes through. To see the policy decision process in a flow chart view, see [Figure 2-6](#) for the J2EE agent view and [Figure 2-7](#) for the web agent view.

For this example, in order to focus on stages of the process most relevant to Policy Agent, certain conditions are assumed as follows:

The user is attempting to access a protected resource after having already accessed a protected resource on the same Domain Name Server (DNS) domain. When the user accessed the first protected resource, OpenSSO Enterprise started a session. The user's attempt to access a second resource, makes this user's session a single sign-on (SSO) session. Therefore, at this point, the following already occurred:

- The user attempted to access a protected resource through a browser (the first resource that the user attempts to access during this session).
- The browser request was intercepted by the agent.
- The browser was redirected to a login uniform resource locator (URL), which is the interface to OpenSSO Enterprise Authentication Service.
- After the user entered valid credentials, the service authenticated the credentials.

The following figure and the corresponding step descriptions demonstrate what occurs after a previously authenticated user attempts to access a second protected resource through a browser. This figure depicts user profiles and policy stored together. Note that these data types are often stored separately.

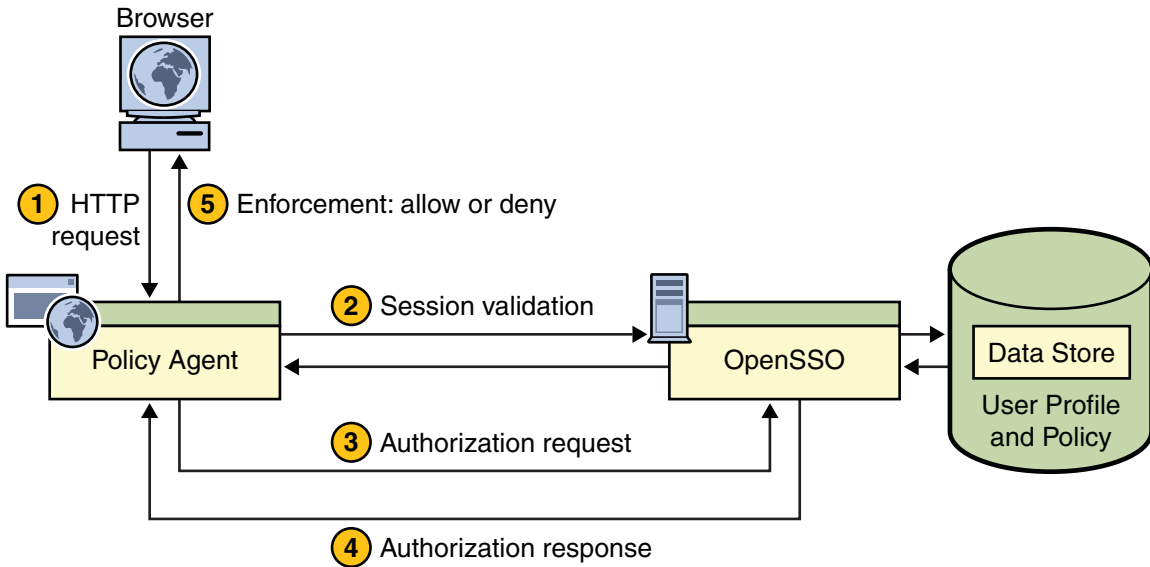


FIGURE 2-5 Policy Agent and the Policy Decision Process

1. The browser sends a request for the protected resource to the deployment container (such as a web or application server) protected by the agent.
2. The agent intercepts the request, checks for a session token embedded in a cookie, and validates the SSO token.

As explained in preceding text, this example assumes that the user's credentials have already been authenticated. Though an SSO session such as this often would *not* require Policy Agent and OpenSSO Enterprise to contact each other during session validation, such contact is sometimes necessary and, therefore, is depicted in [Figure 2-5](#).
3. The agent sends a request to OpenSSO Enterprise Policy Service for user access to the protected resource.
4. OpenSSO Enterprise replies with the policy decision.
5. The agent interprets the policy decision and allows or denies access.

Examples of the Policy Decision Process by Agent Type

This section illustrates the policy decision process through the use of flow charts: one for J2EE agents and one for web agents. These two flow charts show how J2EE agents and web agents can differ in that J2EE security can be enabled for J2EE agents.

Policy Decision Process for J2EE Agents

The figure that follows is a flow chart of the policy decision process for J2EE agents.

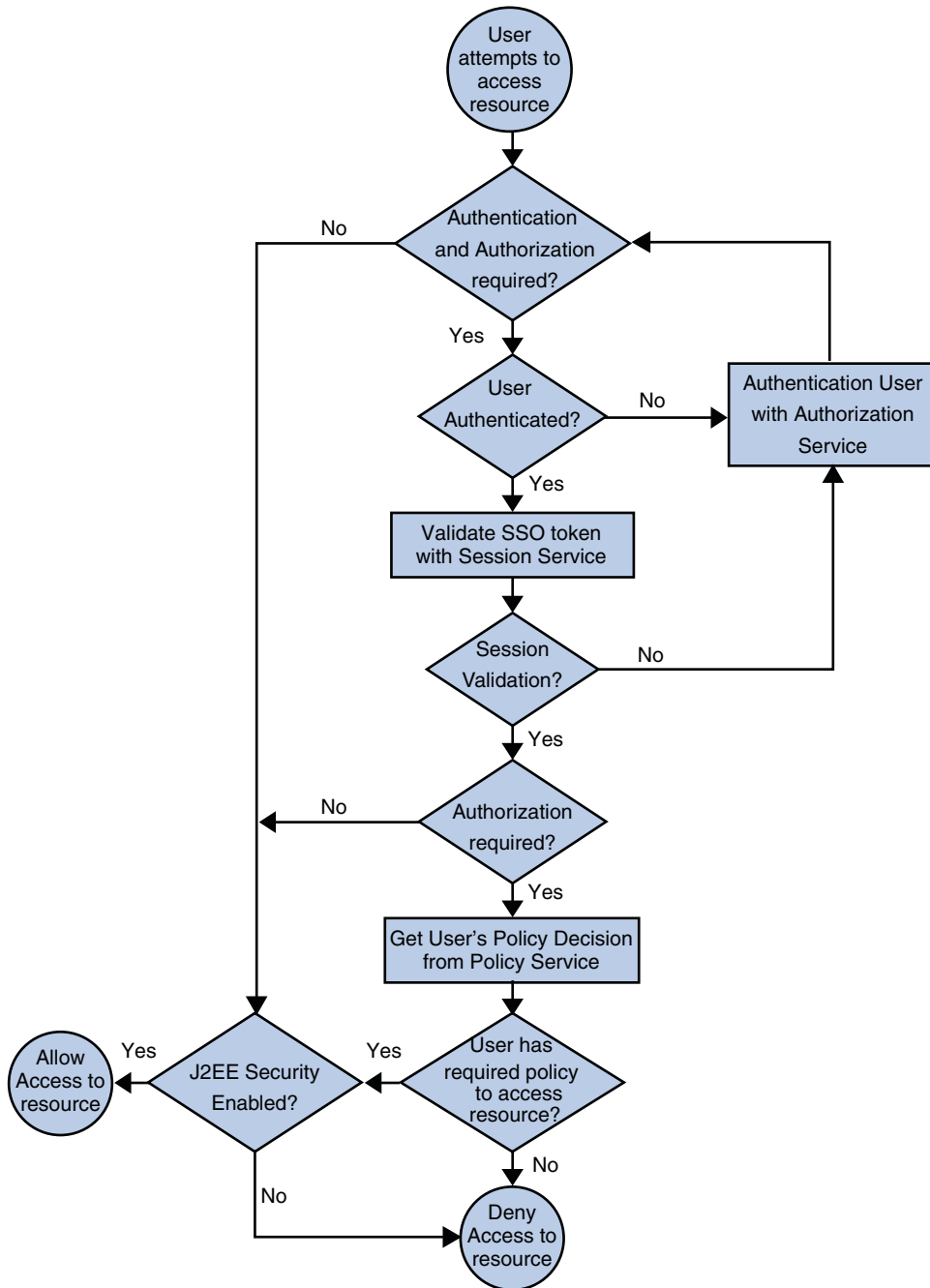


FIGURE 2-6 Policy Agent and the Policy Decision Process

Policy Decision Process for Web Agents

The figure that follows is a flow chart of the policy decision process for web agents.

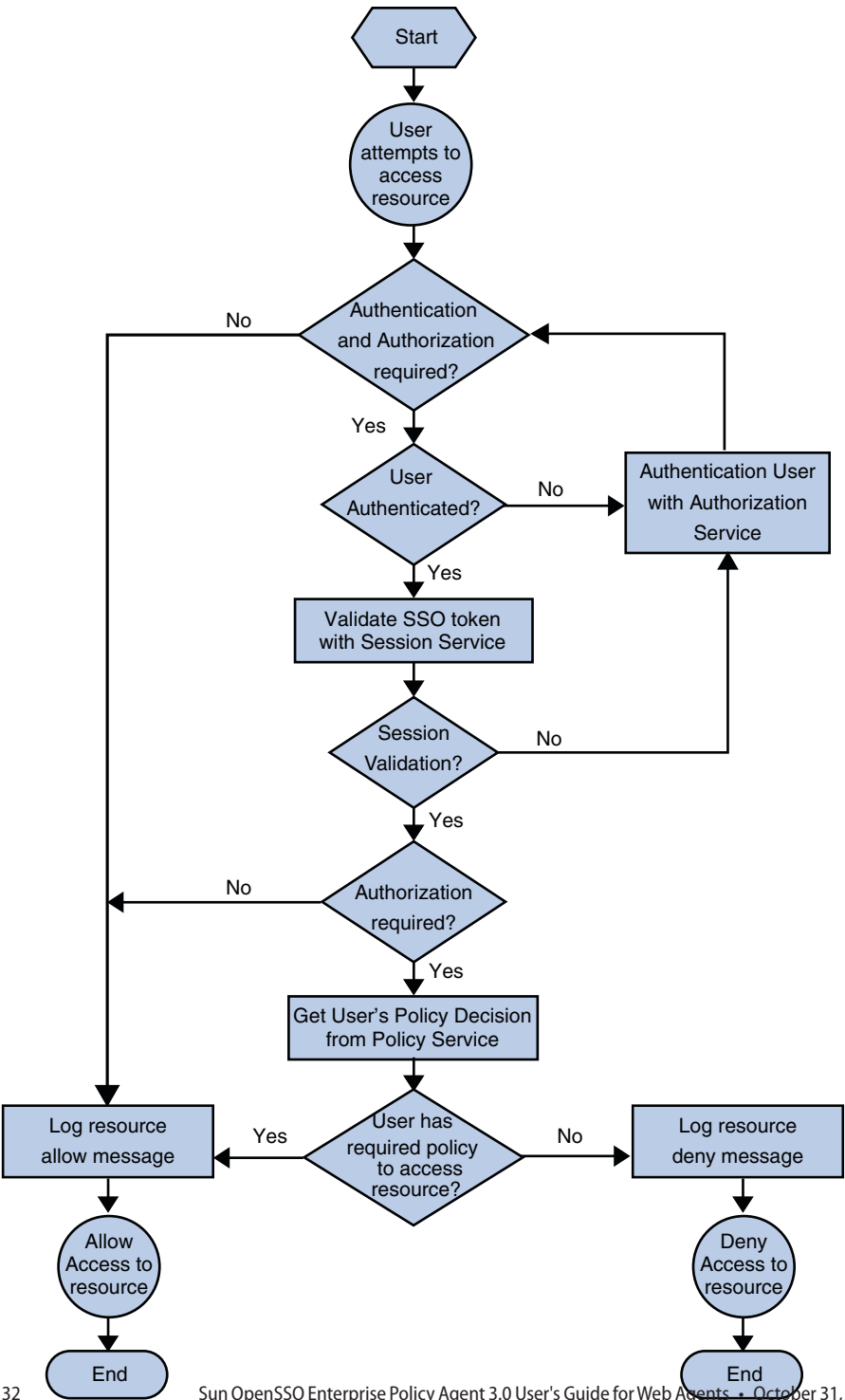


FIGURE 2-7 Policy Agent and the Policy Decision Process

Policy Agent 3.0 and the Centralized Agent Configuration Option

The option to centralize the agent configuration on the OpenSSO Enterprise server, specifically in the central data repository, is new for Policy Agent 3.0. The way this option works is the same for both web agents and J2EE agents. The following figure illustrates the use of this new feature.

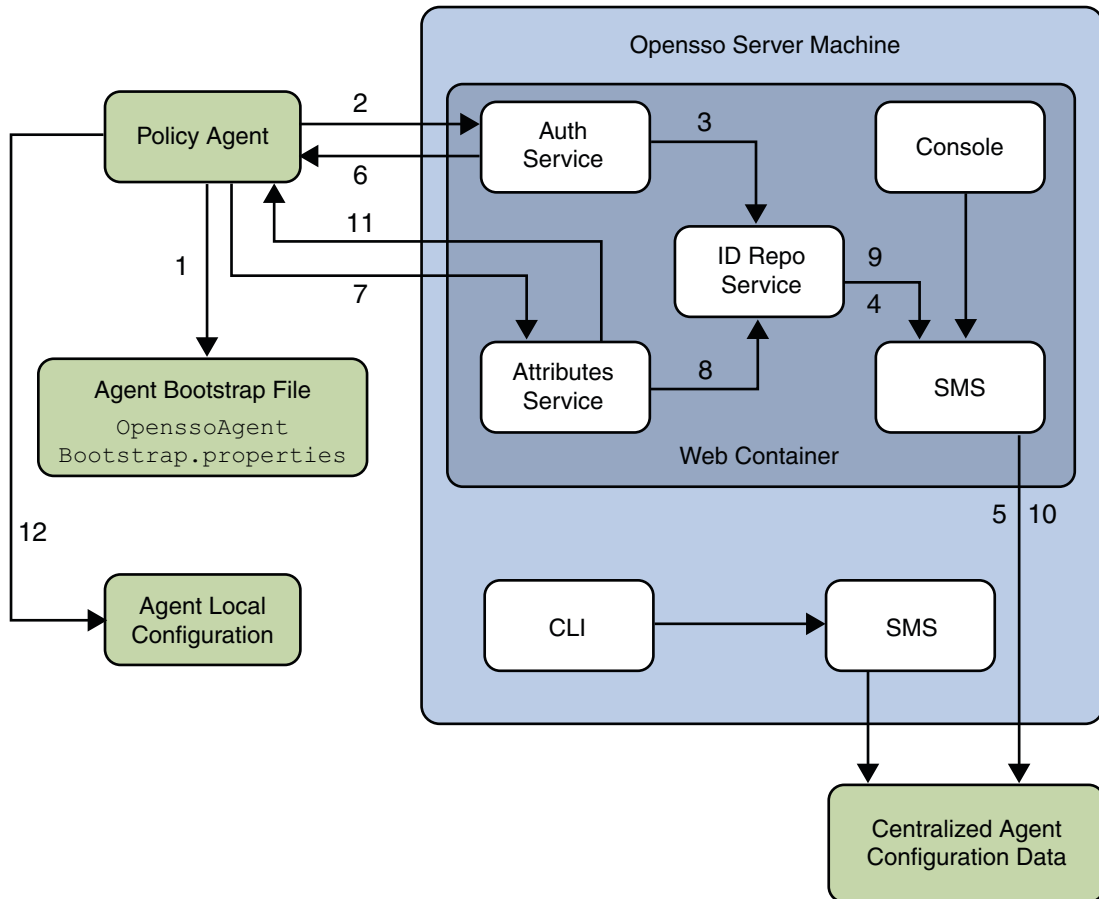


FIGURE 2-8 Centralized Agent Configuration in Policy Agent

Web Agents and J2EE Agents: Similarities and Differences

Both web agents and J2EE agents protect resources hosted on deployment containers (such as web and application servers) or enforce single sign-on with systems that use deployment containers as the front-end in an environment secured by OpenSSO Enterprise. The two types of agents are similar in some ways and yet different in others as outlined in this section.

Web Agents

Web agents control access to content on web servers and proxy servers. The content that web agents can protect include a multitude of services and web resources based on policies configured by an administrator. When a user points a browser to a URL deployed on a protected web or proxy server, the agent intercepts the request and validates the user's session token, if any exists. If the token's authentication level is insufficient (or none exists), the appropriate Authentication Service is called for a login page, prompting the user for (further) authentication. The Authentication Service verifies that the user credentials are valid. For example, the LDAP service verifies that the user name and password are stored in an LDAP v3 compliant directory server, such as Sun Java System Directory Server. After the user's credentials are properly authenticated, the agent examines all the roles and groups (which contain the policies) assigned to the user. Based on the aggregate of all policies assigned to the user, the individual is either allowed or denied access to the URL.

J2EE Agents

OpenSSO Enterprise provides agents for protecting J2EE applications in a variety of deployment containers, such as application and portal servers.

A J2EE policy agent can be installed for protecting a variety of hosted J2EE applications, which might require a varying set of security policy implementation. The security infrastructure of J2EE provides declarative as well as programmatic security that are platform-independent and are supported by all the J2EE-compliant servers. For details on how to use J2EE platform declarative as well as programmatic security, refer to J2EE documentation at <http://java.sun.com/j2ee>.

The agent helps enable role-to-principal mapping for protected J2EE applications with OpenSSO Enterprise principals. Therefore, at runtime, when a J2EE policy is evaluated, the evaluation is against the information available in OpenSSO Enterprise. Using this functionality, you can configure hosted J2EE applications so that they are protected by the J2EE agent, which provides real security services and other key features such as single sign-on. Apart from enabling J2EE security for hosted applications, J2EE agents also provide complete support for OpenSSO Enterprise based URL policies for enforcing access control over web resources hosted in deployment containers, such as an application servers.

While web agents and J2EE agents both work with OpenSSO Enterprise to implement authentication and authorization processes, the design of the J2EE agents allows them to also enforce J2EE security. You can see this difference in terms of enforcing J2EE security by comparing the flow charts of the two agents types: [Figure 2–6](#) and [Figure 2–7](#). The J2EE agents are generally comprised of two components (although this is partially subject to the interfaces exposed and supported by the deployment container): an agent filter for authentication and an agent realm for authorization.

Agent Filter and Authentication

In J2EE agents, the agent filter component manages authentication. The agent filter is a servlet filter, which is supported starting with J2EE 1.3. The agent filter intercepts an inbound request to the server. It checks the request to see if it contains a session token. If one is available, the agent filter validates the token using the OpenSSO Enterprise Session Service. If no token is available, the browser is redirected to the Authentication Service as in a typical SSO exchange. Once the user credentials are authenticated, the request is directed back to the server where the agent filter once again intercepts it, and then validates the newly acquired token. After the user's credentials are validated, the filter enforces J2EE policies or fine-grained URL policies on the resource the user is trying to access. Through this mechanism, the agent filter ensures that only requests with a valid OpenSSO Enterprise token are allowed to access a protected application.

Agent Realm and Authorization

In J2EE agents, the agent realm component manages authorization. A *realm* is a means for a J2EE-compliant application server to provide information about users, groups, and access control to applications deployed on it. It is a scope over which security policy is defined and enforced.

The server is configured to use a specific realm for validation of users and their roles, when attempts are made to access resources. By default, many application servers ship with a number of realm implementations, including the default File Based as well as LDAP, NT, UNIX, and Relational Database Management System (RDBMS). The agent realm component implements the server's realm interface, and allows user and role information to be managed by the OpenSSO Enterprise deployment. The agent realm component makes it possible to provide granular role-based authorization of J2EE resources to users who have been authenticated by the agent filter component.

Key Similarities of the Two Agent Types

The section [“A Generalized Example of the Policy Decision Process”](#) on [page 23](#) describes a deployment that emphasizes the similar tasks performed by web agents and J2EE agents. The two agent types share various other features and tasks that are *not* described in that section. Though this section describes similarities of the two agent types, the features and tasks that they have in common tend to have some differences. However, those differences are often subtle. The details about agent features and tasks are not provided in this section. For details about the

features and tasks for J2EE agents, see. For details about the features and tasks for web agents, see. A list of key features and tasks that web agents and J2EE agents have in common follows along with an explanation of each item:

- [“Configuration Properties” on page 36](#)
- [“Policy Agent Log Files” on page 36](#)
- [“Not-Enforced Lists” on page 36](#)
- [“Personal Profile Attributes and Session Attributes” on page 37](#)

Configuration Properties

All agent configurations have a `OpenSSOAgentBootstrap.properties` file. Regardless of the configuration type, local or centralized, a small subset of properties that are required for the agent to start up and initialize itself are stored in the bootstrap file locally on the server where the agent is installed.

When the agents are installed using a centralized configuration (an option available starting with Policy Agent 3.0), both agent types allow the configuration properties to be configured using the OpenSSO Enterprise Console or the `ssoadm` command-line utility. If the agents are configured locally on the agent host, then for both agent types, the properties that are not stored in the bootstrap file must be configured by editing the `OpenSSOAgentConfiguration.properties` file.

Remark 2–3 Reviewer

I'm unclear about the reach of the `ssoadm` command-line utility. Can it be used to edit properties in the bootstrap file?

For the local configuration scenario, the `OpenSSOAgentConfiguration.properties` files differ slightly between web agents and J2EE agents. The biggest difference between the two agent types is that J2EE agents have extra constructs such as map constructs and list constructs. Configuration properties that are present in the files for both agent types tend to be very similar in terms of functionality.

Policy Agent Log Files

Web agents and J2EE agents can log access information and diagnostic information to an agent log file. Each agent has its own log file, a flat file located on the same host system as the agent. The log file size is configurable. When the active log file reaches the size limit, the log is rotated, which means that the older log information is moved and stored in another log file.

Furthermore, both agent types are capable of logging access information to an OpenSSO Enterprise log file or database table.

Not-Enforced Lists

Both agent types support not-enforced lists. These lists allow for the regular authentication and authorization processes to be bypassed. Two types of not-enforced lists exist: a not-enforced URL list and a not-enforced IP Address list.

A not-enforced URL list is a list of URLs that are not protected by an agent. A resource represented by a URL on a not-enforced URL list is widely available, without restrictions. This list can be set to have a reverse meaning. With a reverse meaning, only URLs on the list are protected. All other URLs are not protected.

A not-enforced IP Address list is a list of IP addresses that are automatically allowed access to resources. When a user is using a computer that has an IP address on the not-enforced IP address list, that user is allowed access to all the resources protected by the respective agent.

Personal Profile Attributes and Session Attributes

Both agent types can fetch and pass along personal profile attributes and session attributes. Client applications protected by an agent can then use information from these attributes to personalize content for the user.

Remark 2–4
Reviewer I suppose that I should mention policy response attributes here, no?

Key Differences Between the Two Agent Types

Many differences exist between J2EE agents and web agents in the way they perform tasks. However, the basic tasks they perform are similar. While the primary purpose of both types of agents is to enforce authentication and authorization before a user can access a protected resource, the two agent types differ in the kind of resources that they can protect and in the way they enforce such policy decisions.

Differences in Protected Resources

Web agents are capable of protecting resources that can be hosted on the web or proxy servers on which they are installed. This protection includes any resource that can be represented as a uniform resource identifier (URI) available on the protected server. Such a protected URI can be resolved by the server to static content files such as HTML files or dynamic content generation programs such as CGI scripts or servlets hosted by an embedded servlet engine. In other words, before a request is evaluated by the web or proxy server, the web agent can evaluate the necessary credentials of a user and can allow or deny access for the requested resource. Once the request is granted access to the resource, it can be processed internally by the web or proxy server as applicable. In other words, the web agent uses the request URL to enforce all policy decisions regardless of what that URL maps to internally in the web server. In cases where the request URL maps to a servlet which in turn invokes other servlets or JSPs, the web agent will not intercept these subsequent resource requests unless such invocation involves a client-side redirect.

A J2EE agent is capable of protecting web and enterprise applications hosted by the application or portal server on which it is installed. These applications may include resources such as HTML pages, servlets, JSP, and Enterprise JavaBeans (EJB). Apart from these resources, any resource that can be accessed as a URI within a protected web application can also be secured by

such agents. For example, images that are packaged within a web application can also be protected by the J2EE Policy Agent. These agents allow the evaluation of J2EE policies and can also enforce OpenSSO Enterprise based URL policies like a web agent on the resources being requested by the user. Minimally the enforcement is done at the outermost requested URL, but can also be done on any intermediate URLs being chained to this resource on most application servers.

Default Scope of Protection

When installed, a web agent automatically protects the entire set of resources available on the web server. However, in order to protect resources within a web application hosted on an application server, the web application must be configured to use the J2EE agent. Thus if multiple web applications are hosted on an application server on which a J2EE agent has been installed, only the web applications that have been specifically configured to use the J2EE agent will be protected by the agent. Other applications will remain unprotected and can potentially malfunction if they depend upon any J2EE security mechanism.

Further, the J2EE agent can only protect resources that are packaged within a web or enterprise application. Certain application servers provide an embedded web server that can be used to host non-packaged web content such as HTML files and images. Such content cannot be protected by a J2EE agent unless it is redeployed as a part of a web application.

Modes of Operation

J2EE agents provide more modes of operation than do web agents. These modes are basically methods for evaluating and enforcing access to resources. You can set the mode according to your site's security requirements. For example, the SSO_ONLY mode is a relatively non-restrictive mode. This mode uses only OpenSSO Enterprise Authentication Service to authenticate users who attempt to access a protected resource.

Some of the modes such as SSO_ONLY and URL_POLICY are also achievable with web agents, whereas other modes of operation such as J2EE_POLICY and ALL modes do not apply to web agents.

For both J2EE agents and web agents, the modes can be set in the OpenSSO Enterprise Console.

In the J2EE_POLICY and ALL modes of operation, J2EE agents enforce J2EE declarative policies as applicable for the protected application and also provide support for evaluation of programmatic security APIs available within J2EE specifications.

Focusing on Web Agents in the Policy Agent 3.0 Release

This guide focuses on web agents. Therefore, this section provides more information about how web agents function generally.

Uses of Web Agents

Web agents function with OpenSSO Enterprise to protect content on web servers and web proxy servers from unauthorized intrusions. They control access to services and web resources based on the policies configured by an administrator. Web agents perform these tasks while providing single sign-on (SSO) and cross domain single sign-on (CDSSO) capabilities as well as URL protection.

Web agents are installed on deployment containers for a variety of reasons. Here are three examples:

- A web agent on a human resources server prevents non-human resources personnel from viewing confidential salary information and other sensitive data.
- A web agent on an operations deployment container allows only network administrators to view network status reports or to modify network administration records.
- A web agent on an engineering deployment container allows authorized personnel from many internal segments of a company to publish and share research and development information. At the same time, the web agent restricts external partners from gaining access to the proprietary information.

In each of these situations, a system administrator must set up policies that allow or deny users access to content on a deployment container. For information on setting policies and for assigning roles and policies to users, see the [Sun Java System Access Manager 7.1 Administration Guide](#).

How Web Agents Work

When a user points a browser to a particular URL on a protected deployment container, a variety of interactions take place as explained in the following numbered list. See the terminology list immediately following this numbered list for a description of terms.

1. The web agent intercepts the request and checks information in the request against not-enforced lists. If specific criteria are met, the authentication process is by passed and access is granted to the resource.
2. If authentication is required, the web agent validates the existing authentication credentials. If the existing authentication level is insufficient, the appropriate OpenSSO Enterprise Authentication Service will present a login page. The login page prompts the user for credentials such as username and password.

3. The authentication service verifies that the user credentials are valid. For example, the default LDAP authentication service verifies that the username and password are stored in Sun Java System Directory Server. You might use other authentication modules such as RADIUS and Certificate modules. In such cases, credentials are not verified by Directory Server but are verified by the appropriate authentication module.
4. If the user's credentials are properly authenticated, the web agent checks if the users is authorized to access the resource.
5. Based on the aggregate of all policies assigned to the user, the individual is either allowed or denied access to the URL.

Terminology: How Web Agents Work

Authentication Level	The ability to access resources can be divided into levels. Therefore, different resources on a deployment container (such as a web server or proxy server) might require different levels of authentication
Service	OpenSSO Enterprise is made of many components. A service is a certain type of component that performs specific tasks. Some of the OpenSSO Enterprise services available are Authentication Service, Naming Service, Session Service, Logging Service, and Policy Service.
Authentication Module	An authentication interface, also referred to as an authentication module, is used to authenticate a user on OpenSSO Enterprise.
Roles	Roles are a Directory Server entry mechanism. A role's members are LDAP entries that possess the role.
Policy	A policy defines rules that specify access privileges to protected resources on a deployment container, such as a web server.

Vital Installation Information for a Web Agent in Policy Agent 3.0

To facilitate the installation and configuration of a web agent in Policy Agent 3.0, essential information is provided in this chapter.

The information for this chapter is presented in the following sections:

- [“Format of the Distribution Files for Web Agents in Policy Agent 3.0” on page 41](#)
- [“Introduction of the agentadmin Program in Web Agents for Policy Agent 3.0” on page 41](#)
- [“Web Agent Directory Structure in Policy Agent 3.0” on page 50](#)
- [“Policy Agent 3.0: Web Agent Properties” on page 53](#)

Though this chapter applies to all web agents developed through the OpenSSO project, occasionally a specific web agent, Policy Agent 3.0 for Sun Java System Web Server 7.0, is used for example purposes. These examples are provided to illustrate general format. Replace web agent specific information where necessary.

Format of the Distribution Files for Web Agents in Policy Agent 3.0

The distribution files for agents are only available as a .zip file, regardless of the platform on which it is to be deployed.

Introduction of the agentadmin Program in Web Agents for Policy Agent 3.0

The agentadmin program is a required installation and uninstallation tool for most web agents in the Policy Agent 3.0 release. For these agents, the most basic of tasks, such as installation and uninstallation can only be performed with this tool.

The location of the agentadmin program is as follows:

PolicyAgent-base/bin

For more information on the location of *PolicyAgent-base*, see [Example 3–17](#).

The following information about the agentadmin program demonstrates the scope of this utility:

- All agent installation and uninstallation is achieved with the agentadmin command.
 - All tasks performed by the agentadmin program, except those involving uninstallation, require the acceptance of a license agreement. This agreement is only presented the first time you use the program.
 - The following table lists options that can be used with the agentadmin command and gives a brief description of the specific task performed with each option.
- A detailed explanation of each option follows the table.

Note – In this section, the options described are the agentadmin program options that apply specifically to web agents.

TABLE 3–1 The agentadmin Program: Supported Options

Option	Task Performed
--install	Installs a new agent instance
--uninstall	Uninstalls an existing Agent instance
--listAgents	Displays details of all the configured agents
--agentInfo	Displays details of the agent corresponding to the specified agent IDs
--version	Displays the version information
--uninstallAll	Uninstalls all agent instances
--usage	Displays the usage message
--help	Displays a brief help message

agentadmin --install

This section demonstrates the format and use of the agentadmin command with the --install option.

EXAMPLE 3-1 Command Format: agentadmin --install

The following example illustrates the format of the agentadmin command with the --install option:

```
./agentadmin --install [--useResponse] [--saveResponse] filename
```

The following arguments are supported with the agentadmin command when using the --install option:

- | | |
|-----------------|---|
| --saveResponse | Use this argument to save all supplied responses to a state file, or response file, represented as <i>filename</i> in command examples. The response file, or state file, can then be used for silent installations. |
| --useResponse | Use this argument to install a web agent in silent mode as all installer prompts are answered with responses previously saved to a response file, represented as <i>filename</i> in command examples. When this argument is used, the installer runs in non-interactive mode. At which time, user interaction is not required. |
| <i>filename</i> | Use this argument to specify the name of a file that will be created as part of the processing of this command. This file stores your responses when this argument is used in conjunction with the --saveResponse argument and provides your responses when this argument is used in conjunction with the --useResponse argument. |

EXAMPLE 3-2 Command Usage: agentadmin --install

When you issue the agentadmin command, you can choose the --install option. With the --install option, you can choose the --saveResponse argument, which requires a file name be provided. The following example illustrates this command when the file name is myfile:

```
./agentadmin --install --saveResponse myfile
```

Once the installer has executed the preceding command successfully, the responses are stored in a state file that can be used for later runs of the installer.

If desired, you can modify the state file and configure the second installation with a different set of configuration parameters.

Then you can issue another command that uses the ./agentadmin --install command and the name of the file that you just created with the --saveResponse argument. The difference between the previous command and this command is that this command uses the --useResponse argument instead of the --saveResponse argument. The following example illustrates this command:

EXAMPLE 3-2 Command Usage: agentadmin --install (Continued)

```
./agentadmin --install --useResponse myfile
```

With this command, the installation prompts run the installer in silent mode, registering all debug messages in the install logs directory.

agentadmin --uninstall

This section demonstrates the format and use of the agentadmin command with the --uninstall option.

EXAMPLE 3-3 Command Format: agentadmin --uninstall

The following example illustrates the format of the agentadmin command with the --uninstall option:

```
./agentadmin --uninstall [--useResponse] [--saveResponse] filename
```

The following arguments are supported with the agentadmin command when using the --uninstall option:

- | | |
|-----------------------|--|
| --saveResponse | Use this argument to save all supplied responses to a state file, or response file, represented as <i>filename</i> in command examples. The response file, or state file, can then be used for silent uninstallations. |
| --useResponse | Use this argument to uninstall a web agent in silent mode as all uninstaller prompts are answered with responses previously saved to a response file, represented as <i>filename</i> in command examples. When this argument is used, the uninstaller runs in non-interactive mode. At which time, user interaction is not required. |
| <i>filename</i> | Use this argument to specify the name of a file that will be created as part of the processing of this command. This file stores your responses when this argument is used in conjunction with the --saveResponse argument and provides your responses when this argument is used in conjunction with the --useResponse argument. |

EXAMPLE 3-4 Command Usage: agentadmin --uninstall

When you issue the agentadmin command, you can choose the --uninstall option. With the --uninstall option, you can choose the --saveResponse argument, which requires a file name be provided. The following example illustrates this command where the file name is myfile:

EXAMPLE 3-4 Command Usage: `agentadmin --uninstall` (Continued)

```
./agentadmin --uninstall --saveResponse myfile
```

Once the uninstaller has executed the preceding command successfully, the responses are stored in a state file that can be used for later runs of the uninstaller.

If desired, you can modify the state file and configure the second uninstallation with a different set of configuration parameters.

Then you can issue another command that uses the `./agentadmin --uninstall` command and the name of the file that you just created with the `--saveResponse` argument. The difference between the previous command and this command is that this command uses the `--useResponse` argument instead of the `--saveResponse` argument. The following example illustrates this command:

```
./agentadmin --uninstall --useResponse myfile
```

With this command, the uninstallation prompts run the uninstaller in silent mode, registering all debug messages in the install logs directory.

agentadmin --listAgents

This section demonstrates the format and use of the `agentadmin` command with the `--listAgents` option.

EXAMPLE 3-5 Command Format: `agentadmin --listAgents`

The following example illustrates the format of the `agentadmin` command with the `--listAgents` option:

```
./agentadmin --listAgents
```

No arguments are currently supported with the `agentadmin` command when using the `--listAgents` option.

EXAMPLE 3-6 Command Usage: `agentadmin --listAgents`

Issuing the `agentadmin` command with the `--listAgents` option provides you with information about all the configured web agents on that machine. For example, if two web agents were configured on Sun Java System Web Server 7.0, the following text demonstrates the type of output that would result from issuing this command:

EXAMPLE 3-6 Command Usage: agentadmin --listAgents (Continued)

The following agents are configured on this Web Server.

The following are the details for agent Agent_001 :-
Sun Java System Web Server Config Directory:
/var/opt/SUNWwbsvr7/https-agentHost1.example.com/config

The following are the details for agent Agent_002 :-
Sun Java System Web Server Config Directory:
/var/opt/SUNWwbsvr7/https-agentHost1.example.com/config

Notice that the agentadmin program provides unique names, such as Agent_001 and Agent_002, to all the web agents that protect the same instance of a deployment container, in this case Web Server 7.0. Each name uniquely identifies the web agent instance.

Note – The string “Agent” in Agent_00x is configurable. You can change this string by editing the following file: *PolicyAgent-base/config/AMToolsConfig.properties*

agentadmin --agentInfo

This section demonstrates the format and use of the agentadmin command with the --agentInfo option.

EXAMPLE 3-7 Command Format: agentadmin --agentInfo

The following example illustrates the format of the agentadmin command with the --agentInfo option:

```
./agentadmin --agentInfo AgentInstance-Dir
```

The following argument is supported with the agentadmin command when using the --agentInfo option:

<i>AgentInstance-Dir</i>	Use this option to specify which agent instance directory, therefore which agent instance such as Agent_002, you are requesting information about.
--------------------------	--

EXAMPLE 3-8 Command Usage: agentadmin --agentInfo

Issuing the agentadmin command with the --agentInfo option provides you with information on the web agent instance that you name in the command. For example, if you want

EXAMPLE 3-8 Command Usage: agentadmin --agentInfo (Continued)

information about a web agent instance named Agent_002 configured on Sun Java System Web Server 7.0, you can issue the command illustrated in the following example to obtain the type of output that follows:

```
./agentadmin --agentInfo Agent_002
```

The following are the details for agent Agent_002 :-
Sun Java System Web Server Config Directory:
/var/opt/SUNWwbsvr7/https-agentHost1.example.com/config

In the preceding example, notice that information is provided only for the agent instance, Agent_002, named in the command.

agentadmin --version

This section demonstrates the format and use of the agentadmin command with the --version option.

EXAMPLE 3-9 Command Format: agentadmin --version

The following example illustrates the format of the agentadmin command with the --version option:

```
./agentadmin --version
```

No arguments are currently supported with the agentadmin command when using the --version option.

EXAMPLE 3-10 Command Usage: agentadmin --version

Issuing the agentadmin command with the --version option provides you with version information for the configured web agents on that machine.

agentadmin --uninstallAll

This section demonstrates the format and use of the agentadmin command with the --uninstallAll option.

EXAMPLE 3-11 Command Format: agentadmin --uninstallAll

The following example illustrates the format of the agentadmin command with the --uninstallAll option:

```
./agentadmin --uninstallAll
```

No arguments are currently supported with the agentadmin command when using the --uninstallAll option.

EXAMPLE 3-12 Command Usage: agentadmin --uninstallAll

Issuing the agentadmin command with the --uninstallAll option runs the agent uninstaller in an iterative mode, enabling you to remove select web agent instances or all web agent instances. You can exit the recursive uninstallation process at any time.

The advantage of this option is that you do not have to remember the details of each installation-related configuration. The agentadmin program provides you with an easy method for displaying every instance of a web agent. You can then decide, case by case, to remove a web agent instance or not.

agentadmin --usage

This section demonstrates the format and use of the agentadmin command with the --usage option.

EXAMPLE 3-13 Command Format: agentadmin --usage

The following example illustrates the format of the agentadmin command with the --usage option:

```
./agentadmin --usage
```

No arguments are currently supported with the agentadmin command when using the --usage option.

EXAMPLE 3-14 Command Usage: agentadmin --usage

Issuing the agentadmin command with the --usage option provides you with a list of the options available with the agentadmin program and a short explanation of each option. The following text is the output you receive after issuing this command:

```
./agentadmin --usage
```


EXAMPLE 3-14 Command Usage: agentadmin --usage (Continued)

Usage: agentadmin <option> [<arguments>]

The available options are:

- install: Installs a new Agent instance.
- uninstall: Uninstalls an existing Agent instance.
- version: Displays the version information.
- uninstallAll: Uninstalls all the agent instances.
- listAgents: Displays details of all the configured agents.
- agentInfo: Displays details of the agent corresponding to the specified agent ID.
- usage: Display the usage message.
- help: Displays a brief help message.

The preceding output serves as the content for the table of agentadmin options, introduced at the beginning of this section.

agentadmin --help

This section demonstrates the format and use of the agentadmin command with the --help option.

EXAMPLE 3-15 Command Format: agentadmin --help

The following example illustrates the format of the agentadmin command with the --help option:

```
./agentadmin --help
```

No arguments are currently supported with the agentadmin command when using the --help option.

EXAMPLE 3-16 Command Usage: agentadmin --help

Issuing the agentadmin command with the --help option provides similar results to issuing the agentadmin command with the --usage option. Both commands provide the same explanations for the options they list. With the --usage option, all agentadmin command options are explained. With the --help option, explanations are not provided for the --usage option or for the --help option itself.

A another difference is that the --help option also provides information about the format of each option while the --usage option does not.

Web Agent Directory Structure in Policy Agent 3.0

The Policy Agent installation directory is referred to as the Policy Agent base directory (or *PolicyAgent-base* in code examples). The location of this directory and its internal structure are important facts that are described in this section.

Location of the Web Agent Base Directory in Policy Agent 3.0

Unpacking the web agent binaries creates a directory named `web_agents`, within which an agent-specific directory is created. Therefore, this directory name is different for each agent. For example, `sjsws_agent` is the directory name applicable to Agent for Sun Java System Web Server.

This agent-specific directory is the Policy Agent base directory, referred to throughout this guide as the *PolicyAgent-base* directory. For the full path to the *PolicyAgent-base* directory, see [Example 3-17](#), which follows.

EXAMPLE 3-17 Policy Agent Base Directory

The directory you choose in which to unpack the web agent binaries is referred to here as *Agent-HomeDirectory*. For illustration purposes, the following example uses Policy Agent 3.0 for Sun Java System Web Server 7.0 to demonstrate the path to the *PolicyAgent-base*. Be aware, that the directory `sjsws_agent` is only an example. This directory will vary from agent to agent:

Agent-HomeDirectory/web_agents/sjsws_agent

References in this book to the *PolicyAgent-base* directory are references to the preceding path.

Inside the Web Agent Base Directory in Policy Agent 3.0

After you finish installing an agent by issuing the `agentadmin -install` command and interacting with the installer, you will need to access web agent files in order to configure and otherwise work with the product. Within the Policy Agent base directory are various subdirectories that contain all agent configuration and log files. The structure of the Policy Agent base directory for a web agent developed through the OpenSSO project is illustrated in [Table 3-2](#).

The list that follows the table provides information about many of the items in the example Policy Agent base directory. The Policy Agent base directory is represented in code examples as *PolicyAgent-base*. The full path to any item in this directory is as follows:

PolicyAgent-base/item-name

where *item-name* represents the name of a file or subdirectory. For example, the full path to the *bin* directory is as follows:

PolicyAgent-base/bin

TABLE 3-2 Example of Policy Agent Base Directory for a Web Agent

Directory Contents: Files and Subdirectories	
license.txt	etc
README	lib
bin	locale
config	logs
data	Agent_001

The preceding example of *PolicyAgent-base* lists files and directories you are likely to find in this directory. The notable items in this directory are summarized in the list that follows:

- bin

This directory contains the `agentadmin` script for the agent bits. You will use this script a great deal. For details about the tasks performed with this script, see [“Introduction of the agentadmin Program in Web Agents for Policy Agent 3.0”](#) on page 41.
- logs

This directory contains installation-related log files, for example log files created after you issue the `agentadmin` command.

Log information is stored in the installation log file after you install a web agent instance. The following is the location of this log file:
PolicyAgent-base/logs/audit/install.log
- lib

The `lib` directory has a list of all the agent libraries that are used by the installer as well as the agent run time.
- locale

This directory has all the agent installer information as well as agent run time specific locale information pertaining to the specific agent.
- data

This directory has all the installer specific data.



Caution – Do not edit any of the files in the data directory under any circumstance. If this directory or any of its content loses data integrity, the agentadmin program cannot function normally.

Agent_001 The full path for this directory is as follows:

PolicyAgent-base/AgentInstance-Dir

where *AgentInstance-Dir* refers to an agent instance directory, which in this case is Agent_001.

Note – This directory does not exist until you successfully install the first instance of a web agent. Once you have successfully executed one run of the agentadmin --install command, an agent specific directory, Agent_00x is created in the Policy Agent base directory. This directory is uniquely tied to an instance of the deployment container, such as a web server instance. Depending on the number of times the agentadmin --install command is run, the number that replaces the x in the Agent_00x directory name will vary.

Furthermore, the string “Agent” in Agent_00x is configurable. You can change this string by editing the following file:

PolicyAgent-base/config/AMToolsConfig.properties

After you successfully install the first instance of a web agent, an agent instance directory named Agent_001 appears in the Policy Agent base directory. The path to this directory is as follows:

PolicyAgent-base/Agent_001

The next installation of the agent creates an agent instance directory named Agent_002. The directories for uninstalled agents are not automatically removed. Therefore, if Agent_001 and Agent_002 are uninstalled, the next agent instance directory is Agent_003.

Agent instance directories contain directories named config and logs.

Note – When a web agent is uninstalled, the config directory is removed from the agent instance directory but the logs directory still exists.

The following table is an example of the contents of an agent instance, such as Agent_001, directory.

Example of an Agent Instance (Agent_001) Directory	
logs	
config	
logs	Two subdirectories exist within this directory as follows: <div><div>audit</div><div>This directory contains the local audit trail for the agent instance.</div><div>debug</div><div>This directory has all the agent-specific debug information. When the agent runs in full debug mode, this directory stores all the debug files that are generated by the agent code.</div></div>
config	This directory contains the OpenSSOAgentBootstrap.properties file and the OpenSSOAgentConfiguration.properties file, which are specific to the agent instance. The OpenSSOAgentBootstrap.properties file applies regardless of the agent configuration: centralized in the OpenSSO Enterprise server or local to the agent. However, the OpenSSOAgentConfiguration.properties file is only meaningful when an agent instance is configured locally. In that scenario, the OpenSSOAgentConfiguration.properties file holds the key to the agent behavior at runtime.

Remark 3-1
Reviewer

Is the above description accurate?

Policy Agent 3.0: Web Agent Properties

The web agent properties changed names from the release 2.2 to the 3.0 release. This section lists those name changes. Furthermore, when applicable, this section provides the property label used with the property names. In prior releases, only property names were used for the properties. However, in Policy Agent 3.0 you can centralize the properties on the OpenSSO Enterprise Console, where labels are more useful.

Web Agent Properties in the OpenSSOAgentBootstrap.properties File

The properties listed in the table that follows can be configured by accessing the OpenSSOAgentBootstrap.properties file. The property names have changed for the 3.0 release as indicated in the table. Labels are not assigned for the properties in this file.

Former Web Agent Property Name	Web Agent 3.0 Property Name
com.sun.am.naming.url	com.sun.identity.agents.config.naming.url
com.sun.am.log.level	com.sun.identity.agents.config.log.level
com.sun.am.policy.agents.config.local.log.file	com.sun.identity.agents.config.local.logfile
com.sun.am.policy.am.username	com.sun.identity.agents.config.username
com.sun.am.policy.am.password	com.sun.identity.agents.config.password
com.sun.am.sslcert.dir	com.sun.identity.agents.config.sslcert.dir
com.sun.am.certdb.prefix	com.sun.identity.agents.config.certdb.prefix
com.sun.am.certdb.password	com.sun.identity.agents.config.certdb.password
com.sun.am.auth.certificate.alias	com.sun.identity.agents.config.certificate.alias
com.sun.am.trust_server_certs	com.sun.identity.agents.config.trust.server.certs
com.sun.am.receive_timeout	com.sun.identity.agents.config.receive.timeout
com.sun.am.connect_timeout	com.sun.identity.agents.config.connect.timeout
com.sun.am.tcp_nodelay.enable	com.sun.identity.agents.config.tcp.nodelay.enable

Web Agent Properties Available Using the OpenSSO Enterprise Console or Other Methods

For the most part, all the properties listed in the various tables that follow can be configured using any of a few different methods: through the OpenSSO Enterprise Console (a centralized agent configuration option), through the ssoadm command line (a centralized agent configuration option), and by editing the OpenSSOAgentConfiguration.properties file (the local agent configuration option). The property names have changed for the 3.0 release as indicated in the various tables. Labels are associated with most of these properties, as indicated. The labels are most useful when using the Console.

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.am.login.url	com.sun.identity.agents.config.login.url Label: OpenSSO Login URL
com.sun.am.cookie.name	com.sun.identity.agents.config.cookie.name Label: Cookie Name
com.sun.am.cookie.secure	com.sun.identity.agents.config.cookie.secure Label: Cookie Security
com.sun.am.policy.agents.config.local.log.rotate	com.sun.identity.agents.config.local.log.rotate Label: Rotate Local Audit Log
com.sun.am.policy.agents.config.local.log.size	com.sun.identity.agents.config.local.log.size Label: Local Audit Log Rotation Size
com.sun.am.policy.agents.config.audit.accesstype	com.sun.identity.agents.config.audit.accesstype Label: Audit Access Types
com.sun.am.policy.agents.config.remote.log	com.sun.identity.agents.config.remote.logfile Label: Remote Log Filename
com.sun.am.policy.agents.config.deny_on_log_failure	com.sun.identity.agents.config.deny.access.log.failure [Remark 3–2 Reviewer: What's with this property? It's not in the Console. Is it still valid?] Label:
com.sun.am.notification.enable	com.sun.identity.agents.config.notification.enable Label: Enable Notifications
com.sun.am.policy.am.url_comparison.case_ignore	com.sun.identity.agents.config.url.comparison.case.ignore Label: URL Comparison Case Sensitivity Check

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.am.polling.interval	com.sun.identity.agents.config.policy.cache.polling.interval Label: Policy Cache Polling Period
com.sun.am.sso.polling.period	com.sun.identity.agents.config.sso.cache.polling.interval Label: SSO Cache Polling Period

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.am.userid.param	com.sun.identity.agents.config.userid.param Label: User ID Parameter
com.sun.am.policy.am.userid.param.type	com.sun.identity.agents.config.userid.param.type Label: User ID Parameter Type
com.sun.am.policy.agents.config. profile.attribute.fetch.mode	com.sun.identity.agents.config. profile.attribute.fetch.mode Label: Profile Attribute Fetch Mode
com.sun.am.policy.agents.config.profile.attribute.map	com.sun.identity.agents.config. profile.attribute.mapping Label: Profile Attribute Mapping
com.sun.am.policy.agents.config. session.attribute.fetch.mode	com.sun.identity.agents.config.session. attribute.fetch.mode Label: Session Attribute Fetch Mode
com.sun.am.policy.agents.config.session.attribute.map	com.sun.identity.agents.config. session.attribute.mapping Label: Session Attribute Mapping
com.sun.am.policy.agents.config. response.attribute.fetch.mode	com.sun.identity.agents.config. response.attribute.fetch.mode Label: Response Attribute Fetch Mode
com.sun.am.policy.agents.config.response.attribute.map	com.sun.identity.agents.config. response.attribute.mapping Label: Response Attribute Mapping

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.load_balancer.enable	com.sun.identity.agents.config.load.balancer.enable Label: Load Balancer Setup
com.sun.am.policy.agents.config.agenturi.prefix	com.sun.identity.agents.config.agenturi.prefix Label: Agent Deploymet URI Prefix
com.sun.am.policy.agents.config.locale	com.sun.identity.agents.config.locale Label: Agent Locale
com.sun.am.policy.agents.config.do_sso_only	com.sun.identity.agents.config.sso.only Label: SSO Only

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
<code>com.sun.am.policy.agents.config.accessdenied.url</code>	<code>com.sun.identity.agents.config.access.denied.url</code> Label: Resources Access Denied URL
<code>com.sun.am.policy.agents.config.fqdn.check.enable</code>	<code>com.sun.identity.agents.config.fqdn.check.enable</code> Label: FQDN Check
<code>com.sun.am.policy.agents.config.fqdn.default</code>	<code>com.sun.identity.agents.config.fqdn.default</code> Label: FQDN Default
<code>com.sun.am.policy.agents.config.fqdn.map</code>	<code>com.sun.identity.agents.config.fqdn.mapping</code> Label: FQDN Virtual Host Map
<code>com.sun.am.policy.agents.config.cookie.reset.enable</code>	<code>com.sun.identity.agents.config.cookie.reset.enable</code> Label: Cookie Reset
<code>com.sun.am.policy.agents.config.cookie.reset.list</code>	<code>com.sun.identity.agents.config.cookie.reset</code> Label: Cookies Reset Name List

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
<code>com.sun.am.policy.agents.config.cookie.domain.list</code>	<code>com.sun.identity.agents.config.cookie.domain</code> Label: Cookies Domain List
<code>com.sun.am.policy.agents.config.anonymous_user</code>	<code>com.sun.identity.agents.config.anonymous.user.id</code> Label: Anonymous User Default Value
<code>com.sun.am.policy.agents.config.anonymous_user.enable</code>	<code>com.sun.identity.agents.config.anonymous.user.enable</code> Label: Anonymous User
<code>com.sun.am.policy.agents.config.notenforced_list</code>	<code>com.sun.identity.agents.config.notenforced.url</code> Label: Not Enforced URLs
<code>com.sun.am.policy.agents.config. notenforced_list.invert</code>	<code>com.sun.identity.agents.config.notenforced.url.invert</code> Label: Invert Check for Not Enforced URLs
<code>com.sun.am.policy.agents.config. notenforced_client_ip_list</code>	<code>com.sun.identity.agents.config.notenforced.ip</code> Label: Not Enforced Client IP List
<code>com.sun.am.policy.agents.config. ignore_policy_evaluation_if_notenforced</code>	<code>com.sun.identity.agents.config. notenforced.url.attributes.enable</code> Label: Fetch Attributes for Notenforced URLs

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.agents.config.postdata.preserve.enable	com.sun.identity.agents.config.postdata.preserve.enable Label: POST Data Preservation
com.sun.am.policy.agents.config.postcache.entry.lifetime	com.sun.identity.agents.config.postcache.entry.lifetime Label: POST Data Entries Cache Period
com.sun.am.policy.agents.config.client_ip_validation.enable	com.sun.identity.agents.config.client.ip.validation.enable Label: Client IP Validation

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.agents.config.profile.attribute.cookie.prefix	com.sun.identity.agents.config.profile.attribute.cookie.prefix Label: Profile Attributes Cookie Prefix
com.sun.am.policy.agents.config.profile.attribute.cookie.maxage	com.sun.identity.agents.config.profile.attribute.cookie.maxage Label: Profile Attributes Cookie Maxage
com.sun.am.policy.agents.config.cdsso.enable	com.sun.identity.agents.config.cdsso.enable Label: Cross Domain SSO
com.sun.am.policy.agents.config.cdcervlet.url	com.sun.identity.agents.config.cdsso.cdcervlet.url Label: CDSSO Servlet URL
com.sun.am.policy.agents.config.logout.url	com.sun.identity.agents.config.logout.url Label: OpenSSO Logout URL
com.sun.am.policy.agents.config.logout.cookie.reset.list	com.sun.identity.agents.config.logout.cookie.reset Label: Logout Cookies List for Reset
com.sun.am.policy.am.fetch_from_root_resource	com.sun.identity.agents.config.fetch.from.root.resource Label: Fetch Policies from Root Resource
com.sun.am.policy.agents.config.get_client_host_name	com.sun.identity.agents.config.get.client.host.name Label: Retrieve Client Hostname
com.sun.am.policy.agents.config.convert_mbyte.enable	com.sun.identity.agents.config.convert.mbyte.enable Label: Native Encoding of Profile Attributes

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.agents.config.encode_url_special_chars.enable	com.sun.identity.agents.config.encode.url.special.chars.enable Label: Encode URL's Special Characters

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.agents.config.ignore_path_info	com.sun.identity.agents.config.ignore.path.info Label: Ignore Path Info in Request URL
com.sun.am.policy.agents.config.override_protocol	com.sun.identity.agents.config.override.protocol Label: Override Request URL Protocol
com.sun.am.policy.agents.config.override_host	com.sun.identity.agents.config.override.host Label: Override Request URL Host
com.sun.am.policy.agents.config.override_port	com.sun.identity.agents.config.override.port Label: Override Request URL Port
com.sun.am.policy.agents.config.override_notification.url	com.sun.identity.agents.config.override.notification.url Label: Override Notification URL
com.sun.am.policy.agents.config.connection_timeout	com.sun.identity.agents.config.connection.timeout [Remark 3–3 Reviewer: What's with this property? It isn't available from the Console.] Label:
com.sun.am.ignore_server_check	com.sun.identity.agents.config.ignore.server.check Label: Ignore Server Check
com.sun.am.poll_primary_server	com.sun.identity.agents.config.poll.primary.server Label: Polling Period for Primary Server
com.sun.am.ignore.preferred_naming_url	com.sun.identity.agents.config.ignore.preferred.naming.url Label: Ignore Preferred Naming URL in Naming Request
com.sun.am.policy.agents.config.proxy.override_host_port	com.sun.identity.agents.config.proxy.override.host.port Label: Override Proxy Server's Host and Port

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.agents.config.domino.check_name_database	com.sun.identity.agents.config.domino.check.name.database Label: Check User in Domino Database

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.agents.config.iis.auth_type	com.sun.identity.agents.config.iis.auth.type Label: Authentication Type
com.sun.am.replaypasswd.key	com.sun.identity.agents.config.replaypasswd.key Label: Replay Password Key
com.sun.am.policy.agents.config.iis.filter_priority	com.sun.identity.agents.config.iis.filter.priority Label: Filter Priority
com.sun.am.policy.agents.config.iis.owa_enabled	com.sun.identity.agents.config.iis.owa.enable Label: Filter configured with OWA
com.sun.am.policy.agents.config.iis.owa_enabled_change_protocol	com.sun.identity.agents.config.iis.owa.enable.change.protocol Label: Change URL Protocol to https
com.sun.am.policy.agents.config.iis.owa_enabled_session_timeout_url	com.sun.identity.agents.config.iis.owa.enable.session.timeout.url Label: Idle Session Timeout Page URL
NEW	com.sun.identity.agents.config.repository.location [Remark 3–4 Reviewer: What's with this property? It isn't available from the Console.] Label:
NEW	com.sun.identity.agents.config.freeformproperties Label: Custom Properties
NEW	com.sun.identity.agents.config.polling.interval Label: Configuration Reload Interval
NEW	com.sun.identity.agents.config.cleanup.interval Label: Configuration Cleanup Interval

Common Web Agent Tasks and Features in Policy Agent 3.0

After installing the web agent and performing the required post-installation steps, you must adjust the agent configuration to your site's specific deployment. This chapter describes how to modify web agents generally. Therefore, the information in this chapter tends to apply to all web agents in the Policy Agent 3.0 software set.

Interaction with Policy Agent 3.0 is enabled to a great extent by configuring the agent properties. However, some interaction with the agent is performed using the `agentadmin` command as explained in [“Introduction of the `agentadmin` Program in Web Agents for Policy Agent 3.0” on page 41](#).

This section focuses on features that involve setting web agent property values. Assigning values to properties is the key method available for configuring agent features. The topics described in this section are typically those of greatest interest in real-world deployment scenarios. This section does not cover every property. For a list of every web agent property, see [“Policy Agent 3.0: Web Agent Properties” on page 53](#)

The manner in which these properties are set varies depending on if the agent configuration is centralized on the OpenSSO Enterprise server or contained locally with the agent. However, regardless of if the agent configuration is local or centralized, a small subset of properties is always stored locally with the agent in the `OpenSSOAgentBootstrap.properties` file. The properties in the bootstrap file are required for the agent to start up and initialize itself. For example, the agent profile name and password used to access the OpenSSO Enterprise server are stored in this bootstrap file. To change the values in the bootstrap file, you must edit the file directly or use the `ssoadm` utility tool. For information about the `ssoadm` utility, see [Appendix D, “Using the `ssoadm` Command-Line Utility With Agents.”](#)

Remark 4–1 **Above, I say one can set properties in the bootstrap file using the `ssoadm` utility. Is that true?**
Reviewer

In terms of the properties *not* stored in the `OpenSSOAgentBootstrap.properties` file and when the J2EE agent configuration is centralized on the OpenSSO Enterprise server, you can use the OpenSSO Enterprise Console or the `ssoadm` command-line utility to set the J2EE agent properties.

When the J2EE agent configuration is local, the Console is not available for agent configuration. Instead, you must set the J2EE agent properties using the `OpenSSOAgentConfiguration.properties` file.



Caution – The content of the `OpenSSOAgentBootstrap.properties` file and the `OpenSSOAgentConfiguration.properties` file are very sensitive. Changes made can result in changes in how the agent works. Errors made can cause the agent to malfunction.

The following is the location of the `OpenSSOAgentBootstrap.properties` file and the `OpenSSOAgentConfiguration.properties` file:

PolicyAgent-base/AgentInstance-Dir/config

For more information about the Policy Agent 3.0 directory structure, see [“Web Agent Directory Structure in Policy Agent 3.0” on page 50](#).

The following topics are discussed in this section:

- [“Providing Failover Protection for a Web Agent” on page 63](#)
- [“Changing the Web Agent Caching Behavior” on page 63](#)
- [“Configuring the Not-Enforced URL List” on page 65](#)
- [“Configuring the Not-Enforced IP Address List” on page 66](#)
- [“Enforcing Authentication Only” on page 66](#)
- [“Providing Personalization Capabilities” on page 66](#)
- [“Setting the Fully Qualified Domain Name” on page 70](#)
- [“Resetting Cookies” on page 72](#)
- [“Configuring CDSSO” on page 72](#)
- [“Setting the REMOTE_USER Server Variable” on page 73](#)
- [“Setting Anonymous User” on page 74](#)
- [“Validating Client IP Addresses” on page 74](#)
- [“Resetting the Agent Profile Password” on page 74](#)
- [“Configuring Web Agent Log Rotation” on page 77](#)
- [“Turning Off FQDN Mapping” on page 71](#)
- [“Enabling Load Balancing” on page 78](#)
- [“Composite Advice” on page 80](#)
- [“Malicious Header Attributes Automatically Cleared by Agents” on page 80](#)

Providing Failover Protection for a Web Agent

When you install a web agent, you can specify a *failover* or backup deployment container, such as a web server, for running OpenSSO Enterprise. This is essentially a high availability option. It ensures that if the deployment container that runs OpenSSO Enterprise service becomes unavailable, the web agent still processes access requests through a secondary, or failover, deployment container running OpenSSO Enterprise service.

Setting up failover protection for the web agent, requires modifying a web agent property. However, you must first install two different instances of OpenSSO Enterprise on two separate deployment containers.

Then follow the instructions about installing the web agent. The web agent installation program prompts you for the host name and port number of the failover deployment container that you have configured to work with OpenSSO Enterprise. The property labeled OpenSSO Login URL (`com.sun.identity.agents.config.login.url`) stores the failover deployment container name.

Set this property in order to store failover server information. Given the values in the following list, the property would be set as shown in [Example 4-1](#).

host1	Name of the primary OpenSSO Enterprise host.
host2	Name of the first failoverOpenSSO Enterprise host.
host3	Name of the second failoverOpenSSO Enterprise host.
example	Name of the domain.
58080	Default port number

EXAMPLE 4-1 Configuration Property Setting for Failover Protection of a Web Agent

```
http://host1.example.com:58080/amserver/UI/Login
http://host2.example.com:58080/amserver/UI/Login
http://host3.example.com:58080/amserver/UI/Login
```

Changing the Web Agent Caching Behavior

Each web agent maintains a cache that stores the policies for every user's session. The cache can be updated by a cache polling mechanism and a cache notification mechanism.

Cache Updates

A web agent maintains a cache of all active sessions involving content that the agent protects. Once an entry is added to an agent's cache, it remains valid for a period of time after which the entry is considered expired and later purged. This feature relies on a polling mechanism.

The web agent property labeled Policy Cache Polling Period (`com.sun.identity.agents.config.policy.cache.polling.interval`) determines the number of minutes an entry will remain in the web agent cache. Once the interval specified by this property has elapsed, the entry is dropped from the cache. By default, the expiration time is set to three minutes.

Hybrid Cache Updates

In this mode, cache entry expiration still applies through use of the polling mechanism. In addition, the web agent gets notified by the OpenSSO Enterprise service about session changes through use of a notification mechanism. Session changes include events such as session logout or a session timeout. When notified of a session or a policy change, the web agent updates the corresponding entry in the cache. Apart from session updates, web agents can also receive policy change updates. Policy changes include events such as updating, deleting, and creating policies.

Web agents have the hybrid cache update mode switched on by default. This is triggered by the web agent property labeled Enable Notifications `com.sun.identity.agents.config.notification.enable`. When this property is disabled, the web agent updates its cache through the polling mechanism only.

Restrictions due to firewalls, as well as the type of deployment container in use, might not allow notifications to work. In such cases, the notification mechanism is turned off.

The web agent sets a timeout period on its cache entries. After its end of life, the cache entry is purged from the web agent's cache. The web agent does not refetch the cache data. The next attempt to access the same entry from cache fails and the web agent makes a round trip to the server and fetches it again to populate the cache. This lazy method of cache updating keeps the web agent cache performing optimally and reduces network traffic.

In a normal deployment situation, policy changes on the server are frequent, which requires sites to accept a certain amount of latency for web agents to reflect policy changes. Each site decides the amount of latency time that is acceptable for the site's specific needs. When setting the Policy Cache Polling Period property, set it to the lower of the two:

- The session idle timeout period
- Your site's accepted latency time for policy changes

Configuring the Not-Enforced URL List

The *not-enforced URL list* defines the resources that should not have any policies (neither allow nor deny) associated with them.

By default, the web agent denies access to all resources on the deployment container that it protects. However, various resources (such as a web site or an application) available through a deployment container might not need to have any policy enforced. Common examples of such resources include the HTML pages and .gif images found in the home pages of web sites and the cascading style sheets (CSS) that apply to these home pages. The user should be able to browse such pages without authenticating. For the home page example, all these resources need to be on the not-enforced URL list or the page will not be displayed properly. The property `com.sun.identity.agents.config.notenforced.url` is used for this purpose. Wild cards can be used to define a pattern of URLs. Space is the separator between the URLs mentioned in the list.

There can be a reverse, or “inverted”, scenario when all the resources on the deployment container, except a list of URLs, are open to any user. In that case, the property `com.sun.identity.agents.config.notenforced.url.invert` would be used to reverse the meaning of `com.sun.identity.agents.config.notenforced.url`. If it is set to `true` (by default it is set to `false`), then the not-enforced URL list would become the enforced list.

EXAMPLE 4-2 Configuration Property Settings for Not-Enforced URL List

The following are examples:

Scenario 1: Not-Enforced URL List

```
com.sun.identity.agents.config.notenforced.url.invert = false
```

```
com.sun.identity.agents.config.notenforced.url =  
http://host1.example.com:80/welcome.html  
http://host1.example.com:80/banner.html
```

In this case, authentication and policies will not be enforced on the two URLs listed in the `notenforcedList`. All other resources will be protected by the web agent.

Scenario 2: Inverted Not-Enforced URL List

```
com.sun.identity.agents.config.notenforced.url.invert = true
```

```
com.sun.identity.agents.config.notenforced.url =  
http://host1.example.com:80/welcome.html  
http://host1.example.com:80/banner.html
```

EXAMPLE 4-2 Configuration Property Settings for Not-Enforced URL List (Continued)

In this case, authentication and policies will be enforced by the web agent on the two URLs mentioned in the `notenforcedList`. All other resources will be accessible to any user.



Caution – If feasible, keep this property set to `false` as such:

```
com.sun.identity.agents.config.notenforced.url.invert = false
```

A value of `false` reduces the chance of unintentionally allowing access to resources.

Configuring the Not-Enforced IP Address List

The `com.sun.identity.agents.config.notenforced.ip` property is used to specify a list of IP addresses. No authentication is required for the requests coming from these client IP addresses.

In other words, the web agent will not enforce policies for the requests originating from the IP addresses in the Not-Enforced IP Address list.

Enforcing Authentication Only

The property `com.sun.identity.agents.config.sso.only` is used to specify if only authentication is enforced for URLs protected by the web agent. If this property is set to `true` (by default it is set to `false`), it indicates that the web agent enforces authentication only, without enforcing policies. After a user logs onto OpenSSO Enterprise successfully, the web agent will not check for policies related to the user and the accessed URLs.

Providing Personalization Capabilities

Web agents in Policy Agent 3.0 can personalize page content for users in three distinct ways as described in the following subsections:

- [“Providing Personalization With Session Attributes” on page 67](#)
- [“Providing Personalization With Policy-Based Response Attributes” on page 68](#)
- [“Providing Personalization With User Profile Attributes Globally” on page 69](#)

Providing Personalization With Session Attributes

Web agents in Policy Agent 3.0 support a feature where a user's session attributes are fetched and set as headers or cookies.

Session attributes are especially effective for transferring information that is dynamic. However, the information transferred only lasts during the current session.

Unlike profile attributes, session attributes are not limited to LDAP attributes retrieved from the user data store. Since session attributes allow non-user profile attributes to be fetched, you can configure the deployment to fetch attributes such as SAML assertion.

The following are examples of session attributes: `UserToken`, `UserId`, `Principal`, `AuthType`, `AuthLevel`, `sun.am.UniversalIdentifier`, `MyProperty`

A good use case for fetching user session attributes presents itself when a post authentication plug-in is involved. A post authentication plug-in performs some tasks right after user authentication. You can configure a post authentication plug-in to fetch data from an external data repository and then set this data as session attributes for that user. These session attributes can be retrieved by the web container and made available to the application.

The web agent property labeled Session Attribute Fetch Mode (`com.sun.identity.agents.config.session.attribute.fetch.mode`) is responsible for fetching session attributes. This property can be configured using the OpenSSO Enterprise Console and can be set to one of the following values:

- NONE
- HTTP_HEADER
- HTTP_COOKIE

When set to `NONE`, no session attributes are fetched and the property labeled Session Attribute Map (`com.sun.identity.agents.config.session.attribute.mapping`) is ignored.

With the Session Attribute Fetch Mode property set to either `HTTP_HEADER` or `HTTP_COOKIE`, the web agent fetches session attributes. Use the following property to configure attributes that are to be forwarded as HTTP headers or cookies:

`com.sun.identity.agents.config.session.attribute.mapping`.

This section illustrates how the Session Attribute Fetch Mode property maps session attributes to headers or cookies.

Session attributes are added to an HTTP header following this format:

```
session_attribute_name|http_header_name[,...]
```

The value of the attribute being fetched in session is `session_attribute_name`. This value gets mapped to a header value as follows: `http_header_name`.

Note – In most cases, in a destination application where `http_header_name` appears as a request header, it is prefixed with `HTTP_` and the following type of conversion takes place:

Lower case letters convert to upper case letters.

Hyphen “-” converts to underscore “_”

“common-name” as an example, converts to “`HTTP_COMMON_NAME`.”

```
com.sun.identity.agents.config.session.attribute.mapping =  
successURL | success-url, contextId | context-id
```

The session attribute is forwarded as a header or a cookie as determined by the end-user applications on the web container that the web agent is protecting. These applications can be considered the consumers of the forwarded header values. The forwarded information is used for the customization and personalization of web pages. You can also write server side plug-ins to put any user session attribute and define the corresponding attribute name and mapping in the preceding property to retrieve the value.

Providing Personalization With Policy-Based Response Attributes

Header attributes can also be determined by OpenSSO Enterprise policy configurations. With policy-based response attributes you can define attribute-value pairs at each policy.

Policy-based response attributes can improve the deployment process, allow greater flexibility in terms of customization, and provide central and hierarchical control of attribute values.

Web agents set policy-based response attributes as headers or cookies based on configuration. All subjects that match this attribute set obtain this attribute. The web agent property labeled Response Attribute Fetch Mode (`com.sun.identity.agents.config.response.attribute.fetch.mode`) controls this functionality:

The default setting for this property is `HTTP_HEADER`. However, this property can be set to any of the following values:

- `NONE`
- `HTTP_HEADER`
- `HTTP_COOKIE`

Attribute mapping is available for response attributes. Therefore, the format of policy information can be mapped to the format of a header or a cookie. The property labeled `Response Attribute Map` (`com.sun.identity.agents.config.response.attribute.mapping`) is used for this type of mapping:

Unlike profile attributes and session attributes, where only the mapped attributes are displayed as headers or cookies, by default, response attributes are set by the agent as headers or cookies based on the setting of the `Response Attribute Fetch Mode` property.

If a response attribute map is specified, then the corresponding attribute mapped name is fetched from the map and its corresponding value is displayed as either a header or a cookie based on the setting of the above property.

Providing Personalization With User Profile Attributes Globally

Web agents in Policy Agent 3.0 have the ability to forward user profile attribute values via HTTP headers to end-web applications. The user profile attribute values come from the server side of OpenSSO Enterprise. The web agent behaves like a broker to obtain and relay user attribute values to the destination servlets, CGI scripts, or ASP pages. These applications can in turn use the attribute values to personalize page content.

This feature is configurable through two web agent properties. To turn this feature on and off, edit the property labeled `Profile Attribute Fetch Mode` (`com.sun.identity.agents.config.profile.attribute.fetch.mode`)

This property can be set to one of the following values:

- NONE
- HTTP_HEADER
- HTTP_COOKIE

When set to `NONE`, the web agent does not fetch LDAP attributes from the server and ignores the web agent property labeled `Profile Attribute Map` (`com.sun.identity.agents.config.profile.attribute.mapping`). In the other two cases, the web agent fetches the attribute.

To configure the attributes that are to be forwarded in the HTTP headers, use the `Profile Attribute Map` property.

By default, some LDAP user attribute names and HTTP header names are set to sample values.

To find the appropriate LDAP user attribute names, check the following XML file on the machine where OpenSSO Enterprise is installed:

ConfigurationDirectory/config/xml/amUser.xml

The attributes in this file could be either OpenSSO Enterprise user attributes or OpenSSO Enterprise dynamic attributes. For an explanation of these two types of user attributes, see [Sun OpenSSO Enterprise 8.0 Administration Guide](#)

The attribute and HTTP header names that need to be forwarded must be determined by the end-user applications on the deployment container that the web agent is protecting. Basically, these applications are the consumers of the forwarded header values (the forwarded information is used for the customization and personalization of web pages).

Setting the Fully Qualified Domain Name

To ensure appropriate user experience, it is necessary that the users access resources protected by the web agent using valid URLs. The configuration property `com.sun.identity.agents.config.fqdn.default` provides the necessary information needed by the web agent to identify if the user is using a valid URL to access the protected resource. If the web agent determines that the incoming request does not have a valid hostname in the URL, it redirects the user to the corresponding URL with a valid hostname. The difference between the redirect URL and the URL originally used by the user is only the hostname, which is changed by the web agent to a fully qualified domain name (FQDN) as per the value specified in this property.

This is a required configuration property without which the deployment container may not start up correctly. This property is set during the web agent installation and must not be modified unless absolutely necessary to accommodate deployment requirements. An invalid value for this property can result in the deployment container becoming unusable or the resources becoming inaccessible.

The property `com.sun.identity.agents.config.fqdn.mapping` provides another way by which the web agent can resolve partial or malformed access URLs and take corrective action. The web agent gives precedence to the entries defined in this property over the value defined in the `com.sun.identity.agents.config.fqdn.default` property. If none of the entries in this property matches the hostname specified in the user request, the agent uses the value specified for `com.sun.identity.agents.config.fqdn.default` property.

The `com.sun.identity.agents.config.fqdn.mapping` property can be used for creating a mapping for more than one hostname. This may be the case when the deployment container protected by this agent is accessible by more than one hostname. However, this feature must be used with caution as it can lead to the deployment container resources becoming inaccessible.

This property can also be used to override the behavior of the web agent in cases where necessary. The format for specifying the property `com.sun.identity.agents.config.fqdn.mapping` is:

```
com.sun.identity.agents.config.fqdn.mapping =
[invalid_hostname|valid_hostname][,...]
```

where:

`invalid_hostname` is a possible invalid hostname such as partial hostname or an IP address that the user may provide .

`valid_hostname` is the corresponding valid hostname that is fully qualified. For example, the following is a possible value specified for hostname `xyz.domain1.com`:

```
com.sun.identity.agents.config.fqdn.mapping = xyz|xyz.domain1.com,
xyz.domain1|xyz.domain1.com
```

This value maps `xyz` and `xyz.domain1` to the FQDN `xyz.domain1.com`.

This property can also be used in such a way that the web agent uses the name specified in this map instead of the deployment container's actual name.

If you want your server to be addressed as `xyz.hostname.com` whereas the actual name of the server is `abc.hostname.com`. The browser only knows `xyz.hostname.com` and you have specified policies using `xyz.hostname.com` in the OpenSSO Enterprise Console. In this file, set the mapping as `com.sun.identity.agents.config.fqdn.mapping = valid|xyz.hostname.com`.

Turning Off FQDN Mapping

You can disable fully qualified domain name (FQDN) mapping of HTTP requests.

This checking capability is controlled by the web agent properties labeled FQDN default (`com.sun.identity.agents.config.fqdn.default`) and FQDN Virtual Host Map.

The web agent property labeled FQDN Check (`com.sun.identity.agents.config.fqdn.check.enable`) is enabled by default. When this property is enabled, the request URLs that are present in user requests are checked against FQDN values. However, you can turn FQDN checking off if you choose. Turning off FQDN checking might be beneficial when a deployment includes a number of virtual servers for which the agent is configured using FQDN mapping.

Remark 4–2
Reviewer Does the explanation above make sense?

Resetting Cookies

The cookie reset feature enables the web agent to reset some cookies in the browser session while redirecting to OpenSSO Enterprise for authentication.

This feature is configurable through two web agent properties: the property labeled Cookie Reset (`com.sun.identity.agents.config.cookie.reset.enable`) and the property labeled Cookies Reset Name List (`com.sun.identity.agents.config.cookie.reset`):

- **Cookie Reset**

This property must be enabled if the web agent is to reset cookies in the response while redirecting to OpenSSO Enterprise for authentication. By default, this property is not enabled.

- **Cookie Reset Name List**

The values for this property are the cookies that need to be reset in the response while redirecting to OpenSSO Enterprise for authentication. This property is used only if the Cookie Reset feature is enabled.

Cookie details must be specified in the following format:

```
name[=value][;Domain=value]
```

For example,

```
cookie1=value1,  
cookie2=value2;Domain=example.com
```

Configuring CDSSO

The cross domain single sign-on (CDSSO) feature is configurable through three web agent properties. Enable or disable this feature with the property labeled Cross Domain SSO (`com.sun.identity.agents.config.cdsso.enable = true`). By default, this property is not enabled, and the feature is turned off.

Set the URL where the CDC controller is installed by assigning the URL as the value to the property labeled CDSSO Servlet URL (`com.sun.identity.agents.config.cdsso.cdcservlet.url`).

The following is an example of the value that could be assigned to this property:

```
http://OpenSSOhost.example.com:58080/amserver/cdcservlet
```

The third property involved in configuring CDSSO is labeled Cookies Domain List (`com.sun.identity.agents.config.cookie.domain`). This property allows you to specify a

list of domains in which cookies have to be set in a CDSSO scenario. This property is used only if CDSSO is enabled. If you leave this property blank, then the fully qualified cookie domain for the web agent server will be used for setting the cookie domain. In such a case, it is a host cookie and not a domain cookie.

For more information on CDSSO, see [Sun OpenSSO Enterprise 8.0 Technical Overview](#).

Setting the REMOTE_USER Server Variable

The property labeled User ID Parameter (`com.sun.identity.agents.config.userid.param`) allows you to configure the user ID parameter passed by the session or user profile information from OpenSSO Enterprise. The user ID value is used by the agent to set the value of the REMOTE_USER server variable. By default, this parameter is set to UserToken and is fetched from session attributes.

It can be set to any other session attribute or profile attribute. Another property determines where to retrieve the value, from user profiles or from session properties.

Remark 4–3 Reviewer

Is the sentence above correct? Also, I don't know what the note below is trying to say. It talks about a value being fetched, but I don't know which value. Anyway, part of the explanation is missing here.

Note – Be aware that when this value is fetched from session properties, you must write server-side plug-in code in order to add session attributes after authentication.

Example 1: This example lists the values that you can set the User ID Parameter property to for session attributes:

SESSION (this is default)

UserToken (UserId, Principal, or any other session attribute)

Example 2: This example lists the values that you can set the User ID Parameter property to for LDAP user profile attributes:

LDAP

cn (any profile attribute)

Setting Anonymous User

For resources on the not-enforced list, the default configuration does not allow the `REMOTE_USER` variable to be set. The not-enforced list refers to values assigned to the property labeled Not Enforced URLs (`com.sun.identity.agents.config.notenforced.url`)

To enable the `REMOTE_USER` variable to be set for not-enforced URLs, you must enable the web agent property labeled Anonymous User (`com.sun.identity.agents.config.anonymous.user.enable`). By default, this value is not enabled.

When you enable this property, the value of `REMOTE_USER` is set to the value contained in the web agent property labeled Anonymous User Default Value (`com.sun.identity.agents.config.anonymous.user.id`). **[Remark 4–4 Reviewer: Is the preceding property accurate? It's in the Deprecated Agent Properties section.]** By default, the value assigned to this property is anonymous.

Validating Client IP Addresses

This feature can be used to enhance security by preventing the stealing or *hijacking* of SSO tokens.

By default, the web agent labeled Client IP Validation (`com.sun.identity.agents.config.client.ip.validation.enable`) is not enabled.

If you enable this property, client IP address validation is enforced for each incoming request that contains an SSO token. If the IP address from which the request was generated does not match the IP address issued for the SSO token, the request is denied. This is essentially the same as enforcing a deny policy.

This feature should not be used, however, if the client browser uses a web proxy or if a load balancer exists somewhere between the client browser and the agent-protected deployment container. In such cases, the IP address appearing in the request will not reflect the real IP address on which the client browser runs.

Resetting the Agent Profile Password

This section describes how to reset the web agent profile password. The web agent stores the agent profile password in an encrypted format in the `OpenSSOAgentBootstrap.properties` file. Specifically, the agent profile password is stored as a value assigned to the following property:

Remark 4–5 The term used before for the agent profile password was "shared secret." I'm assuming that
Reviewer term is obsolete at this point. Am I correct? Should I be calling this the "agent profile password"?

```
com.sun.identity.agents.config.password
```

Agents in the Policy Agent software set must authenticate with the OpenSSO Enterprise server in order for the two components to interact. To authenticate, the agent must provide its name (the agent profile name) and agent profile password. This password was established and encrypted as part of the web agent installation process. However, you can change this password if you choose.

To reset (change) the agent profile password, follow the subsequent platform-specific instructions about using the encryption utility to encrypt the agent profile password and then assign that value as the value to the `com.sun.identity.agents.config.password` property (follow the steps according to the platform on which the agent is installed).

Remark 4–6 I'm not sure if all the stuff in this section still applies. For example, do the scripts such as
Reviewer `crypt_util` still apply? What about the steps in general?

▼ To Reset the Agent Profile Password on Solaris Systems

- 1 Go to the following directory:

```
PolicyAgent-base/bin
```

- 2 Execute the following script in the command line:

```
# ./crypt_util agent-password
```

where *agent-password* represents the agent profile password, that along with the agent user name, allows the web agent to authenticate with OpenSSO Enterprise.

- 3 Copy the output obtained after issuing the `# ./crypt_util agent-password` command and paste it as the value for the following property:

```
com.sun.identity.agents.config.password
```

- 4 Restart the deployment container and try accessing any resource protected by the agent.

If the agent gets redirected to OpenSSO Enterprise, this indicates the above steps were executed properly.

▼ To Reset the Agent Profile Password on Windows Systems

- 1 Go to the following directory:

PolicyAgent-base\bin

- 2 Execute the following script in the command line

cryptit agent-password

where *agent-password* represents the agent profile password, that along with the agent user name, allows the web agent to authenticate with OpenSSO Enterprise.

- 3 Copy the output obtained after issuing the *cryptit agent-password* command and paste it as the value for the following property:

`com.sun.identity.agents.config.password`

- 4 Restart the deployment container and try accessing any resource protected by the agent.

If the agent gets redirected to OpenSSO Enterprise, this indicates the above steps were executed properly.

▼ To Reset the Agent Profile Password on Linux Systems

- 1 Go to the following directory:

PolicyAgent-base/bin

- 2 Execute the following script in the command line:

crypt_util agent-password

where *agent-password* represents the agent profile password, that along with the agent user name, allows the web agent to authenticate with OpenSSO Enterprise.

- 3 Copy the output obtained after issuing the *crypt_util agent-password* command and paste it as the value for the following property:

`com.sun.identity.agents.config.password`

- 4 Restart the deployment container and try accessing any resource protected by the agent.

If the agent gets redirected to OpenSSO Enterprise, this indicates the above steps were executed properly.

Configuring Web Agent Log Rotation

For web agents, when the current log file reaches a specific size, a new log file is created. Log information is then stored in the new log file until it reaches the size limit. This default behavior is configurable. Therefore, log rotation can be turned off and the size limit can be changed.

Note – The following types of information are logged for Policy Agent 3.0:

- Troubleshooting information
- Access denied information
- Access allowed information

The troubleshooting, or diagnostic, information is stored in log files, locally, with the web agent. The access denied and access allowed information, which is often referred to as audit-related information, can be stored both locally and with OpenSSO Enterprise.

Remark 4–7 Reviewer

It seems this might have changed in 3.0. What's missing and/or wrong from this explanation?

Configuration that relates to the local log files is performed by editing the web agent property labeled Rotate Local Audit Log (`com.sun.identity.agents.config.local.log.rotate`). The Rotate Local Audit Log property is accessible using the OpenSSO Enterprise Console. Configuration that relates to the audit related logs stored with OpenSSO Enterprise is not controlled by an agent property, but this type of configuration can also be implemented using the Console.

The log rotation described in this section refers to logs that store troubleshooting information locally.

The local logs are rotated automatically since by default, the Rotate Local Audit Log property is enabled. **[Remark 4–8 Reviewer: Is it true that this property is enabled by default? On my deployment, it is not enabled. Did this change for 3.0?]** When this property is not enabled, no rotation takes place for the local log file.

The following properties are also related to log rotation:

- The value of the following web agent property indicates the location of the debug file:
`com.sun.identity.agents.config.local.logfile`
This property is available in the `OpenSSOAgentBootstrap.properties` file with the agent, not in the Console. Since a local audit file is created during agent installation, the location of that file is assigned to this property at that time.
- The value of the web agent property labeled Local Audit Log Rotation Size (`com.sun.identity.agents.config.local.log.size`) indicates the maximum number of bytes the debug file holds. You can set this agent property in the Console.

This property controls the log file size in that a new log file is created when the current log file reaches a specific size. The file size should be a minimum of 3000 bytes. The default size is 10 megabytes.

**Remark 4–9
Reviewer** **Is this default correct?**

When a new log file is created an index appends to the name of the log file as such:

amAgent-1
amAgent-2

Where *amAgent* represents the fully qualified path name to the log files excluding the appended number. The numbers *1* and *2* represent the appended number. The appended number indicates the chronological order in which information of a given size was filed away into its respective log file. There is no limit to the number of log files that can be rotated.

Enabling Load Balancing

Various web agent properties influence the enablement of load balancing. Edit the properties that apply, according to the location of the load balancer or load balancers in your deployment, as follows:

- [“Load Balancer in Front of OpenSSO Enterprise” on page 78](#)
- [“Load Balancer in Front of the Web Agent” on page 79](#)
- [“Load Balancers in Front of Both the Web Agent and OpenSSO Enterprise” on page 80](#)

Load Balancer in Front of OpenSSO Enterprise

When a load balancer is deployed in front of OpenSSO Enterprise and a web agent interacts with the load balancer, the following web agent properties must be edited:

- `com.sun.identity.agents.config.naming.url` (accessible in the `OpenSSOAgentBootstrap.properties` file)
- OpenSSO Login URL (`com.sun.identity.agents.config.login.url`)
- Load Balancer Setup (`com.sun.identity.agents.config.load.balancer.enable`)

EXAMPLE 4–3 Property Settings: Load Balancer in Front of OpenSSO Enterprise

This example illustrates the web agent property settings that can be used to enable load balancing:

EXAMPLE 4-3 Property Settings: Load Balancer in Front of OpenSSO Enterprise (Continued)

Property (name or label)	Setting
com.sun.identity.agents.config.naming.url (accessible in the OpenSSOAgentBootstrap.properties file)	LB-url /amserver/namingservice
OpenSSO Login URL	LB-url /amserver/UI/Login
Load Balancer Setup	Enabled

where *LB-url* represents the load balancer URL. The following example is a conceivable load balancer URL:

http://LBhost.example.com:8080

Load Balancer in Front of the Web Agent

In many cases, when a load balancer is deployed in front of the web agent only the property labeled FQDN Virtual Host Map (`com.sun.identity.agents.config.fqdn.mapping`) is required.

Assign the following value to the the FQDN Virtual Host Map property:

`valid|LB-hostname`

where *LB-hostname* represents the name of the machine on which the load balancer is located.

However, if SSL-termination or a proxy server is used in the deployment, all the following web agent properties should be set in addition to the preceding property:

- Override Request URL Protocol (`com.sun.identity.agents.config.override.protocol`)
- Override Request URL Host (`com.sun.identity.agents.config.override.host`)
- Override Request URL Port (`com.sun.identity.agents.config.override.port`)
- Override Notification URL (`com.sun.identity.agents.config.agenturi.prefix`)

This example illustrates how properties can be set to enable load balancing when the protocol, hostname, and port number of the load balancer differ from that of the web agent. However, if the load balancer and the web agent share one of these characteristics, such as the protocol or hostname, then the respective property would be left blank instead of being assigned a value of *true*.

Property (name or label)	Setting
Override Request URL Protocol	true
Override Request URL Host	true
Override Request URL Port	true
Override Notification URL	<i>LB-url/amagent</i>

Load Balancers in Front of Both the Web Agent and OpenSSO Enterprise

This scenario is simply a combination of the scenarios described in the preceding sections. See [“Load Balancer in Front of OpenSSO Enterprise” on page 78](#) and [“Load Balancer in Front of the Web Agent” on page 79](#).

Remark 4–10
Reviewer Below are sections on Composite advice and Malicious headers. These sections were in 2.2 as new features. They seem useful for 3.0, but I'm not certain how to incorporate them into this guide. Right now, no tasks are related to these features. I'm not sure if there should be another section, such as "More Features of Web Agents in the Policy Agent 3.0 Software Set." Or maybe these could be incorporated into FAQs. I'm not sure.

Composite Advice

Web agents provide a composite advice feature. This feature allows the policy and authentication services of OpenSSO Enterprise to decouple the advice handling mechanism of the agents. This allows you to introduce and manage custom advices by solely writing OpenSSO Enterprise side plug-ins. You are not required to make changes on the agent side. Such advices are honored automatically by the composite advice handling mechanism.

Malicious Header Attributes Automatically Cleared by Agents

For web agents in the Policy Agent software set, malicious header attributes are automatically cleared. The benefit of this automatic clean up is that security is improved. Header information that is *not* automatically cleared has greater risk of being accessed

Silent Installation and Uninstallation of a Web Agent in Policy Agent 3.0

In addition to a standard installation and uninstallation of web agents, you can perform a silent installation or uninstallation. This appendix provides a description of how to use the silent option for the installation and uninstallation of web agents.

About Silent Installation and Uninstallation of a Web Agent in Policy Agent 3.0

A silent installation or uninstallation refers to installing or uninstalling a program by implementing a script. The script is part of a state file. The script provides all the answers that you would normally supply to the installation or uninstallation program interactively. Running the script saves time and is useful when you want to install or uninstall multiple instances of Policy Agent using the same parameters in each instance.

Silent installation is a simple two-step process of generating a state file and then using that state file. To generate a state file, you record the installation or uninstallation process, entering all the required information that you would enter during a standard installation or uninstallation. Then you run the installation or uninstallation program with the state file as the input source.

Generating a State File for a Web Agent Installation

This section describes how to generate a state file for installing a web agent. This task requires you to issue a command that records the information you will enter as you follow the agent installation steps. Enter all the necessary installation information in order to create a complete state file.

▼ To Generate a State File for a Web Agent Installation

To generate a state file for a web agent installation, perform the following:

1 Change to the following directory:

PolicyAgent-base/bin

This directory contains the `agentadmin` program, which is used for installing a web agent and for performing other tasks. For more information on the `agentadmin` program, see [“Introduction of the agentadmin Program in Web Agents for Policy Agent 3.0” on page 41](#).

2 Issue the following command:

```
./agentadmin --install --saveResponse filename
```

`--saveResponse` An option that saves all of your responses to installation prompts in a state file.

filename Represents the name that you choose for the state file.

3 Answer the prompts to install the agent.

Your answers to the prompts are recorded in the state file. When the installation is complete, the state file is created in the same directory where the installation program is located.

Note – When generated, a state file will have read permissions for all users. However, because the state file contains clear text passwords, it is recommended that you change the file permissions to restrict read and write access to the user root.

Using a State File for a Web Agent Silent Installation

The installation program does not validate inputs or the state in the silent installation. Ensure that the proper environment exists before performing a silent installation.

▼ To Install a Web Agent Using a State File

To perform a silent installation of a web agent using a state file, perform the following:

1 Change to the following directory:

PolicyAgent-base/bin

At this point, this `bin` directory should contain the `agentadmin` program and the web agent installation state file.

2 Issue the following command:

```
./agentadmin --install --useResponse filename
```

- useResponse An option that directs the installer to run in non-interactive mode as it obtains all responses to prompts from the named state file.
- filename* Represents the name of the state file from which the installer obtains all responses.

The installation takes place hidden from view. After completion, the program exits automatically and displays the prompt.

Generating a State File for a Web Agent Uninstallation

This section describes how to generate a state file for uninstalling a web agent. This task requires you to issue a command that records the information you will enter as you follow the agent uninstallation steps. Enter all the necessary uninstallation information in order to create a complete state file.

▼ To Generate a State File for a Web Agent Uninstallation

To generate a state file for uninstallation of a web agent, perform the following:

1 Change to the following directory:

PolicyAgent-base/bin

This directory contains the `agentadmin` program, which is used for uninstalling a web agent and for performing other tasks. For more information on the `agentadmin` program, see [“Introduction of the agentadmin Program in Web Agents for Policy Agent 3.0” on page 41](#).

2 Issue the following command:

```
./agentadmin --uninstall --saveResponse filename
```

- saveResponse An option that saves all of your responses to uninstallation prompts in a state file.

filename Represents the name that you choose for the state file.

3 Answer the prompts to uninstall the agent.

Your answers to the prompts are recorded in the state file. When uninstallation is complete, the state file is created in the same directory where the uninstallation program is located.

Note – When generated, a state file will have read permissions for all users. However, because the state file contains clear text passwords, it is recommended that you change the file permissions to restrict read and write access to the user root.

Using a State File for a Web Agent Silent Uninstallation

The uninstallation program does not validate inputs or the state in the silent installation. Ensure that the proper environment exists before performing a silent uninstallation.

▼ To Uninstall a Web Agent Using a State File

To perform a silent uninstallation of a web agent using a state file, perform the following:

1 Change to the following directory:

PolicyAgent-base/bin

At this point, this bin directory should contain the agentadmin program and the web agent uninstallation state file.

2 Issue the following command:

```
./agentadmin --uninstall --useResponse filename
```

-useResponse An option that runs the uninstallation process in non-interactive mode as all responses to prompts are obtained from the named state file.

filename Represents the name of the state file from which the installer obtains all responses.

The uninstallation takes place hidden from view. After completion, the program exits automatically and displays the prompt.

Wildcard Matching in Policy Agent 3.0 Web Agents

The OpenSSO Enterprise policy service supports policy definitions that use either of the two following wildcards:

**Remark B-1
Reviewer**

Is this write up accurate? Does everything in this wildcard section fit the agent type?

- “The Multi—Level Wildcard: *” on page 86
- “The One-Level Wildcard: -*” on page 87

These wildcards can be used in policy related situations. For example, when using the OpenSSO Enterprise Console or the `ssoadm` utility to create policies or when configuring the Policy Agent property to set the not-enforced list.



Caution – When issuing the `ssoadm` command, if you include values that contain wildcards (* or -*), then the name/value pair should be enclosed in double quotes to avoid substitution by the shell. For more information about the `ssoadm` command, see [Appendix D, “Using the ssoadm Command-Line Utility With Agents.”](#)

For creating a policy, the following are feasible examples of the wildcards in use:

`http://agentHost:8090/agentsample/*` and
`http://agentHost:8090/agentsample/example-*-/example.html`.

For the not-enforced list, the following are feasible examples of the wildcards in use:

`http://agentHost:8090/agentsample.com/*.gif` and
`http://agentHost:8090/agentsample/*-/images`

Note – A policy resource can have either the multi-level wildcard (*) or the one-level wildcard (*-), but not both. Using both types of wildcards in the same policy resource is not supported.

The Multi—Level Wildcard: *

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

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

The following examples show the multi-level wildcard character when used with the forward slash (/) as the delimiter character:

- The asterisk (*) matches zero or more characters, except the question mark, in the resource name, including the forward slash (/). For example, ...B-example/* matches ...B-example/b/c/d, but doesn't match ...B-example/?
- Multiple consecutive forward slash characters (/) do not match with a single forward slash character (/). For example, ...B-example/*/A-example doesn't match ...B-example/A-example.
- Any number of trailing forward slash characters (/) are not recognized as part of the resource name. For example, ...B-example/ or ...B-example// are treated the same as ...B-example.

TABLE B–1 Examples of the Asterisk (*) as the Multi-Level Wildcard

Pattern	Matches	Does Not Match
http://A-example.com:80/*	http://A-example.com:80 http://A-example.com:80/ http://A-example.com:80/index.html http://A-example.com:80/x.gif	http://B-example.com:80/ http://A-example.com:8080/index.html http://A-example.com:80/a?b=1
http://A-example.com:80/*.html	http://A-example.com:80/index.html http://A-example.com:80/pub/ab.html http://A-example.com:80/pri/xy.html	http://A-example.com/index.html http://A-example.com:80/x.gif http://B-example.com/index.html
http://A-example.com:80/*/ab	http://A-example.com:80/pri/xy/ab/xy/ab http://A-example.com:80/xy/ab	http://A-example.com/ab http://A-example.com/ab.html http://B-example.com:80/ab

TABLE B-1 Examples of the Asterisk (*) as the Multi-Level Wildcard (Continued)

Pattern	Matches	Does Not Match
<code>http://A-example.com:80/ab/*de</code>	<code>http://A-example.com:80/ab/123/de</code> <code>http://A-example.com:80/ab/ab/de</code> <code>http://A-example.com:80/ab/de/ab/de</code> <code>http://A-example.com:80/ab//de</code>	<code>http://A-example.com:80/ab/de</code> <code>http://A-example.com:80/ab/de</code> <code>http://B-example.com:80/ab/de/ab/de</code>

The One-Level Wildcard: `-*-`

The one-level wildcard (`-*-`) matches only the defined level starting at the location of the one-level wildcard to the next delimiter boundary. The “defined level” refers to the area between delimiter boundaries. Many of the rules that apply to the multi—level wildcard also apply to the one-level wildcard.

The following list summarizes the behavior of hyphen-asterisk-hyphen (`-*-`) as a wildcard:

- Matches zero or more occurrences of any character except for the forward slash and the question mark (?).
- Does not span across multiple levels in a URL
- Cannot be escaped. Therefore, the backslash character (`\`) or other characters cannot be used to escape the hyphen-asterisk-hyphen, as such `\-*-`.

The following examples show the one-level wildcard when used with the forward slash (/) as the delimiter character:

- The one-level wildcard (`-*-`) matches zero or more characters (except for the forward slash and the question mark) in the resource name. For example, `...B-example/-*-` doesn't match `...B-example/b/c/` or `...B-example/b?`
- Multiple consecutive forward slash characters (/) do not match with a single forward slash character (/). For example, `...B-example/-*-/A-example` doesn't match `...B-example/A-example`.
- Any number of trailing forward slash characters (/) are not recognized as part of the resource name. For example, `...B-example/` or `...B-example//` are treated the same as `...B-example`.

TABLE B-2 Examples of the One—Level Wildcard (-*-)

Pattern	Matches	Does Not Match
http://A-example.com:80/b/-*-	http://A-example.com:80/b http://A-example.com:80/b/ http://A-example.com:80/b/cd/	http://A-example.com:80/b/c?d=e http://A-example.com:80/b/cd/e http://A-example.com:8080/b/
http://A-example.com:80/b/-*-/f	http://A-example.com:80/b/c/f http://A-example.com:80/b/cde/f	http://A-example.com:80/b/c/e/f http://A-example.com:80/f/
http://A-example.com:80/b/c/-*-/f	http://A-example.com:80/b/cde/f http://A-example.com:80/b/cd/f http://A-example.com:80/b/c/f	http://A-example.com:80/b/c/e/f http://A-example.com:80/b/c/ http://A-example.com:80/b/c/fg

Precautions Against Session-Cookie Hijacking in an OpenSSO Enterprise Deployment

This appendix provides information about precautions you can take against specific security threats related to session-cookie hijacking in an OpenSSO Enterprise deployment. A common concern for administrators who want to restrict access to web-based applications in an OpenSSO Enterprise deployment is that hackers might use applications, referred to as “rogue” or “untrusted,” to hijack session cookies. This appendix describes the threat posed by this hacking technique and provides configuration steps you can perform to guard against this threat, as described in the following sections:

- [“Defining Key Cookie Hijacking Security Issues” on page 89](#)
- [“OpenSSO Enterprise Solution: Cookie Hijacking Security Issues” on page 93](#)
- [“Implementing the OpenSSO Enterprise Solution for Cookie Hijacking Security Issues” on page 96](#)

The tasks presented in this document apply to a deployment with the following components:

- Starting with OpenSSO Enterprise 8.0
- Starting with Policy Agent 3.0

Defining Key Cookie Hijacking Security Issues

The tasks presented in this document address specific security issues related to session-cookie hijacking. This section defines those security issues.

The term “cookie hijacking” simply refers to a situation where an impostor (a hacker, perhaps using an untrusted application) gains unauthorized access to cookies. Therefore, cookie hijacking, by itself, does not refer to unauthorized access to protected web resources. When the cookies being hijacked are session cookies, then cookie hijacking potentially increases the threat of unauthorized access to protected web resources, depending upon how the system is configured.

This section provides background information about specific security issues related to session-cookie hijacking, illustrating how this type of cookie hijacking is possible and how it

can potentially lead to unauthorized access of protected web resources. This section provides the underlying basis for understanding how the tasks presented in this document enable OpenSSO Enterprise to handle the specified security issues.

OpenSSO Enterprise provides Single Sign-On (SSO) and Cross Domain Single Sign-On (CDSSO) features, enabling users to seamlessly authenticate across multiple web-based applications. OpenSSO Enterprise is able to provide this seamless authentication by using hypertext transfer protocol (HTTP) or secure hypertext transfer protocol (HTTPS) to set session cookies on users' browsers.

The way OpenSSO Enterprise is configured influences the way it sets the session cookies. Prior to the implementation of the tasks outlined in this document, OpenSSO Enterprise sets session cookies for the entire domain. Therefore, all applications hosted on the domain share the same session cookies. This scenario could enable an untrusted application to intervene and gain access to the session cookies. Specific configuration steps presented in this document address this issue. After you perform the configuration, OpenSSO Enterprise prevents different applications from sharing the same session cookies.

The following list provides some details about possible security issues related to session-cookie hijacking (labels, such as “Security Issue: Insecure Protocol,” are used to make the issues easy to identify and discuss):

Security Issue: Insecure Protocol

An application does not use a secure protocol, such as HTTPS, which makes the session cookie prone to network eavesdropping.

The following security issues could apply if you do not perform the tasks presented in this document.

Security Issue: Shared Session Cookies

All applications share the same HTTP or HTTPS session cookie. This shared session-cookie scenario enables hackers to intervene by using an untrusted application to hijack the session cookie. With the hijacked session cookie, the untrusted application can impersonate the user and access protected web resources.

Security Issue: A Less Secure Application

If a single “less secure” application is hacked, the security of the entire infrastructure is compromised.

Security Issue: Access to User Profile Attributes

The untrusted application can use the session cookie to obtain and possibly modify the profile attributes of the user. If the user has administrative privileges, the application could do much more damage.

[Figure C–1](#) illustrates a typical OpenSSO Enterprise deployment within an enterprise. While the figure helps to define security issues related to cookie hijacking, it also helps to define the solution. Therefore, the figure applies to a deployment where the tasks presented subsequently in this document have already been performed.

The deployment illustrated in the figure uses SSO. Moreover, as part of the solution, the OpenSSO Enterprise implementation of CDSSO is enabled. To guard against potential threats posed by cookie hijacking, CDSSO enablement is required regardless of the number of domains involved. Having CDSSO enabled when the deployment involves a single domain might seem counterintuitive, but ultimately security is increased by taking advantage of the CDSSO framework. For other information about the use of CDSSO in this type of deployment, see [“Enabling OpenSSO Enterprise to Use Unique SSO Tokens” on page 95](#).

The numbers in the figure correspond to the numbered explanations that follow. The explanations combine to provide an outline of the process that takes place when a request is made for a protected resource, emphasizing how the SSO token flows between components. The deployment includes a single virtual AuthN (Authentication) and AuthZ (Policy) server (denoted as OpenSSO Enterprise or Identity Provider in the figure), and a number of applications (Service Providers), denoted as Trusted Application and Untrusted Application in the diagram. The Service Providers are usually front-ended with an agent (specifically, an agent from the Policy Agent 3.0 software set) that handles the SSO (AuthN) and Policy (AuthZ).

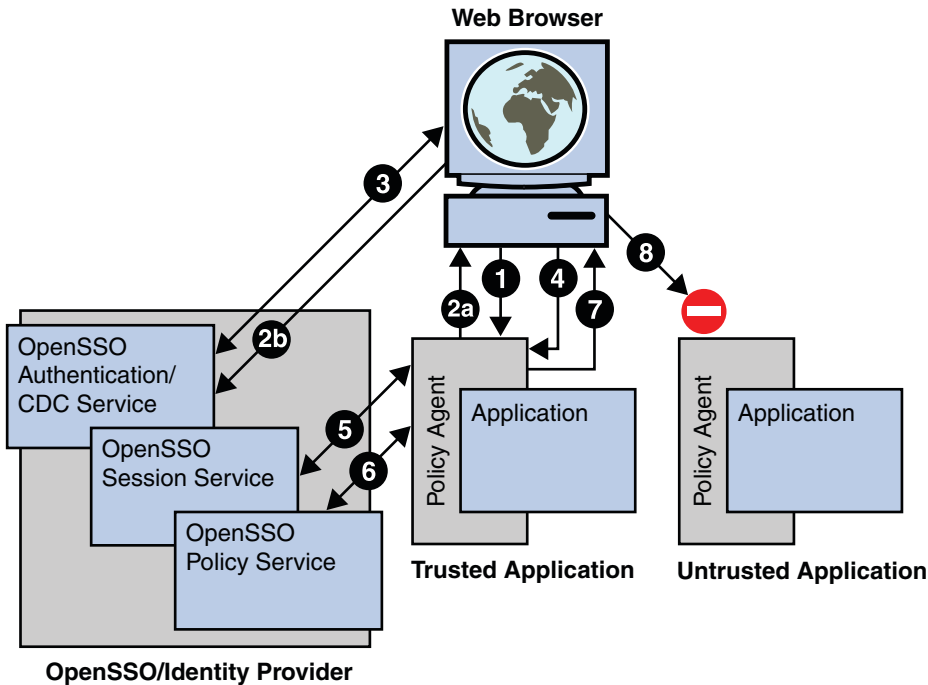


FIGURE C-1 The Role of the Session Cookie in Allowing Access to Protected Web Resources

Note – Figure C-1 does not include every detail involved in the flow of the SSO token. The figure provides a simplified view that corresponds to the scope of this document.

1. The user accesses an application through a web browser. The agent, which is an agent from the Policy Agent 3.0 software set, intercepts the request and checks for the user's privilege to access the application. The check is specifically a check for a valid OpenSSO Enterprise SSO token, which is set as a session cookie in the user's browser.
2. Assuming the user does not have a valid SSO token, the agent redirects the browser to OpenSSO Enterprise Authentication Service. The agent also provides its identity using the Liberty AuthN request specification. Since this single redirection is a two step process, it is illustrated in the figure as two substeps: 2A and 2B.
3. OpenSSO Enterprise Authentication Service authenticates the user and, after successful authentication, redirects the user back to the target application with the SSO token as part of the URL query parameter (in a format specified by Liberty AuthN response specification).
4. The agent receives a successful AuthN response from OpenSSO Enterprise Authentication Service. It gets the SSO token and sets it as a session cookie for the host (rather than for the domain) to the browser's request.

5. The agent validates the SSO token with OpenSSO Enterprise Session Service.
6. After a successful SSO token validation in Step 5, the agent then checks the permissions for the user to access the application with OpenSSO Enterprise Policy Service.
7. If permission is granted in Step 6, the user is allowed access to the application.
8. As indicated in the figure with an incomplete connection to Untrusted Application, the same SSO token cannot be used to gain access to an untrusted application. OpenSSO Enterprise denies such access since the SSO token is unique to the application and may not be shared or reissued to other agents or applications.

OpenSSO Enterprise Solution: Cookie Hijacking Security Issues

This section explains how performing the tasks described in this document enables OpenSSO Enterprise to handle the security issues discussed in the preceding section, [“Defining Key Cookie Hijacking Security Issues” on page 89](#).

Note – When applications use a secure protocol such as HTTPS, the SSO Token is not visible to network snooping. This security issue is labeled “Security Issue: Insecure Protocol” in this document. Ensuring that all protected resources use a secure protocol is not a security measure administered using OpenSSO Enterprise, but this a very prudent security measure that you should consider implementing if it is not currently in place.

OpenSSO Enterprise Solution: Shared Session Cookies

The security issue labeled “Security Issue: Shared Session Cookies” in this document pertains to applications sharing the same HTTP or HTTPS session cookie. OpenSSO Enterprise addresses this security threat by issuing a unique SSO token to each Application/Agent after the user has been authenticated. The unique SSO token is referred to as a “restricted token.”

The term “Application/Agent,” indicates that the restricted token is inextricably connected to the application and to the agent (which specifically refers to an agent from the Policy Agent 3.0 software set). Since each user's SSO token is unique for each Application/Agent, the increased security provided by this scenario prevents an untrusted application, impersonating the user, from accessing other applications. More specifically, since the SSO token (restricted token) assigned to a user (as a part of the user's session) is associated with the agent that did the initial redirection for authentication, all subsequent requests are checked to verify that they are coming from the same agent. Thus, if a hacker tries to use the same restricted token to access another application, a security violation is thrown.

What makes the restricted token “restricted” is not related to the syntax of the token. The syntax of a restricted token is the same as that of a regular SSO token. Instead, a specific constraint is associated with the restricted token. This constraint is what ensures that the restricted token is only used for an application that a given agent protects.

OpenSSO Enterprise Solution: A Less Secure Application

The security issue labeled “Security Issue: A Less Secure Application” in this document pertains to the potential threat of applications that are “less secure.” With the OpenSSO Enterprise solution, if one application is somehow compromised, the hacker cannot hack into other applications.

OpenSSO Enterprise Solution: Modification of Profile Attributes

The security issue labeled “Security Issue: Access to User Profile Attributes” in this document pertains to the threat posed by an untrusted application modifying the profile attributes of the user. The OpenSSO Enterprise solution to this issue does not change the SSO token. The restricted SSO token is identical to the regular SSO token ID. However, the set of Session Service operations that accept restricted SSO token IDs is limited. This functionality enables OpenSSO Enterprise to prevent applications from modifying profile attributes of the user.

Key Aspects of the OpenSSO Enterprise Solution: Cookie Hijacking Security Issues

The following subsections explain some of the key or more complex aspects of the OpenSSO Enterprise solution to the cookie hijacking security issues defined in this document.

The Significance of Creating an Agent Profile for Each Agent

For Policy Agent 3.0, each agent has its own agent profile. This is significant from a security point of view. In fact all the configuration steps presented in this document rely on each agent having its own agent profile. As explained in [“OpenSSO Enterprise Solution: Shared Session Cookies” on page 93](#), a situation where applications share session cookies presents a security issue. OpenSSO Enterprise addresses this potential security risk by issuing a unique SSO token to each agent. In order to issue a unique SSO token to each agent, each agent must have its own agent profile.

OpenSSO Enterprise Session Cookies Involved in Issuing Unique SSO Tokens

When OpenSSO Enterprise is configured to issue unique SSO tokens for each Application/Agent, the following cookies are involved:

Cookie Name	Cookie Value (place holder)	Example Cookie Domain Information
iPlanetDirectoryPro	SSO-token	amHost.example.com

The value of this cookie, which is represented in the preceding table with the place holder *SSO-token*, is the actual value of the token. The domain is set to the host name of the OpenSSO Enterprise instance where the user was authenticated.

Cookie Name	Cookie Value (place holder)	Example Cookie Domain Information
iPlanetDirectoryPro	restricted-SSO-token	agentHost.example.com

The value of this cookie, which is represented in the preceding table with the place holder *restricted-SSO-token*, is the actual value of the token. The domain is set to the host name of the agent instance for which the restricted token is issued.

Cookie Name	Example Cookie Value	Example Cookie Domain Information
sunIdentityServerAuthNServer	https://amHost.example.com:8080	.example.com

The value of this cookie, which is represented in the preceding table with the example URL `https://amHost.example.com:8080`, is the URL of the OpenSSO Enterprise instance where the user was authenticated. The protocol used for this particular example is HTTPS while the port number is a non-default example, 8080. The domain must be set such that it covers all the instances of OpenSSO Enterprise installed on the network.

Enabling OpenSSO Enterprise to Use Unique SSO Tokens

To enable OpenSSO Enterprise to issue unique SSO tokens, you must enable CDSSO. Therefore, though CDSSO is usually enabled for multiple-domain deployments, in this case, CDSSO must be enabled whether the entire deployment is on a single domain or is spread across multiple domains. In no way does enabling CDSSO for a single domain negatively affect the deployment.

The next section describes the steps required to configure OpenSSO Enterprise to prevent session-cookie hijacking from causing a breach of security.

Implementing the OpenSSO Enterprise Solution for Cookie Hijacking Security Issues

The instructions presented in this section provide a solution to the potential risks related to session-cookie hijacking as outlined in this document. This section includes two subsections as follows:

- [“Updating an Agent Profile” on page 96](#)
- [“Configuring the OpenSSO Enterprise Deployment Against Cookie Hijacking” on page 97](#)

This section also provides the other configuration steps necessary to guard web resources in an OpenSSO Enterprise deployment against the threat of session-cookie hijacking.

After you perform the tasks presented in this document, OpenSSO Enterprise starts enforcing restrictions on the sessions it creates. The new configuration enables OpenSSO Enterprise to more closely track aspects of each session. At that point, not only does OpenSSO Enterprise record which agent performed the initial redirection for authentication it also tracks the applications to which an SSO token has been issued. OpenSSO Enterprise uses this information to facilitate the processing of each subsequent request and to prevent unauthorized access to protected web resources.

Updating an Agent Profile

Updating the agent profile is not necessary for implementing the precautions against cookie hijacking. However, instructions are provided in case you would like to change the profile.

▼ To Update the Agent Profile Credentials in the Web Agent

This task description provides alternatives for steps that differ according to platform: UNIX-based systems or Windows systems. For a more thorough explanation about updating the agent profile for a web agent, see the corresponding Policy Agent 3.0 documentation.

1 Update the following property in the `OpenSSOAgentConfiguration.properties`:

`com.sun.identity.agents.config.username`

Replace the value of this property with the agent profile ID (name) you just updated in OpenSSO Enterprise Console. Therefore, this property would then appear similarly to the following:

`com.sun.identity.agents.config.username = agent-profile-ID`

2 Change directories to the following:

- **UNIX-based Systems**

PolicyAgent-base/bin

- **Windows Systems**

PolicyAgent-base\bin

3 Execute the following script in the command line:

- **UNIX-based Systems**

crypt_util agent-profile-password

- **Windows Systems**

cryptit agent-profile-password

where *agent-profile-password* represents the agent profile password you updated in OpenSSO Enterprise Console.

4 Copy the output obtained after issuing the script in the previous step and paste it as the value for the following property:

`com.sun.identity.agents.config.password`

This property would appear similarly to the example that follows, where *encrypted-password-string* is a place holder for the real encrypted password:

`com.sun.identity.agents.config.password = encrypted-password-string`

5 Ensure that protected web resources are accessible.

- a. **Restart the web container.**

- b. **Attempt to access any resource protected by the web agent.**

If the attempt to access a protected resource results in a redirection to OpenSSO Enterprise, then the preceding steps were executed properly.

Configuring the OpenSSO Enterprise Deployment Against Cookie Hijacking

At this point in the configuration, OpenSSO Enterprise has not been configured to associate an SSO token to a specific agent profile. The steps in this section enable this type of association.

Ultimately, the new configuration introduces “restricted tokens” into the OpenSSO Enterprise deployment, guarding against security issues as described in this document.

▼ To Configure the OpenSSO Enterprise Deployment Against Cookie Hijacking

This task description includes configuration information for agents in the Policy Agent 3.0 software set. Perform the task on every agent instance for which you want to enhance security. The best practice is to perform the task on all the agent instances in the OpenSSO Enterprise deployment. As part of the configuration of each agent instance, you must also make specific configurations directly to OpenSSO Enterprise. For this task, be prepared to access the OpenSSO Enterprise Console, the `OpenSSOAgentBootstrap.properties` file, and the `OpenSSOAgentConfiguration.properties` file.

1 Using the OpenSSO Enterprise Console, access the agent profile configuration page.

2 Add the appropriate value to the field labeled Agent Key Value.

Set the agent properties with a key/value pair as illustrated in the example that follows. This property is used by OpenSSO Enterprise to retrieve an agent profile from an agent repository for credential assertions about agents. Currently, only one property is valid. All other properties are ignored. Use the following format:

`agentRootURL=protocol://hostname:port/`

The preceding entry must be precise. Be aware that the string “agentRootURL” is case sensitive. Also, the slash following the port number is required.

protocol Represents the protocol used, such as HTTP or HTTPS.

hostname Represents the host name of the machine on which the agent resides. This machine also hosts the resources that the agent protects.

port Represents the port number on which the agent is installed. The agent listens to incoming traffic on this port and, from the port, intercepts all requests to access resources on the host.

The following is an example of how this property could be set:

`agentRootURL=https://agentHost.example.com:8080/`

3 Edit the `OpenSSOAgentConfiguration.properties` file as described in the substeps that follow:

a. Set the property that enables CDSSO to `true` as illustrated:

`com.sun.identity.agents.config.cdsso.enable = true`

The preceding property setting enables CDSSO, which is required for each agent instance since the agent will use functionality provided by the CDSSO feature.

- b. Set the property that stores the URL users are directed to after they log in successfully in a deployment enabled for CDSSO:

```
com.sun.identity.agents.config.cdsso.cdcervlet.url =  
https://amHost.example.com:8080/amserver/cdcervlet
```

- 4 Restart the container that hosts the agent.
- 5 Edit the appropriate OpenSSO Enterprise properties in the OpenSSO Enterprise Console as described in the substeps that follow:

- a. Set the following property to `true` as illustrated:

```
com.sun.identity.enableUniqueSSOTokenCookie = true
```

- b. Set the following property exactly as it is illustrated:

```
com.sun.identity.authentication.uniqueCookieName =  
sunIdentityServerAuthNServer
```

- c. Set the following property to a domain such that it covers all the OpenSSO Enterprise instances installed:

```
com.sun.identity.authentication.uniqueCookieDomain
```

The following example illustrates how this property would be set if the domain name was `example.com`.

```
com.sun.identity.authentication.uniqueCookieDomain = .example.com
```

- 6 In the OpenSSO Enterprise Console, select the Configuration tab.
- 7 Scroll as needed to the System Properties list and click Platform.
- 8 In the Cookie Domain list, change the cookie domain name.

This step enables OpenSSO Enterprise to set host-specific session cookies instead of domain-wide session cookies.

- a. Ensure that the default domain, such as “`example.com`,” is selected.
- b. Click Remove.
- c. Enter the name of the machine hosting the OpenSSO Enterprise instance.
For example:
`amHost.example.com`
- d. Click Add.

- 9 **Ensure that the proper cookies appear in a browser.**
 - a. **Use a browser to access a resource that is protected by the agent that you just configured.**
 - b. **Check the browser's cookie settings to ensure that the three following cookies appear:**

Cookie Name	Example Cookie Value	Example Cookie Domain Information
iPlanetDirectoryPro	<i>SSO-token</i>	amHost.example.com
iPlanetDirectoryPro	<i>restricted-SSO-token</i>	agentHost.example.com
sunIdentityServerAuthNServer	https://amHost.example.com:8080	.example.com

For more information about the preceding cookies, see [“OpenSSO Enterprise Session Cookies Involved in Issuing Unique SSO Tokens”](#) on page 95.

Using the ssoadm Command-Line Utility With Agents

When the agent configuration is centralized, you can configure OpenSSO Enterprise through the OpenSSO Enterprise Console or through the command line, using the ssoadm utility.

Note – The ssoadm utility cannot be used in scenarios where the agent configuration is stored locally with the agent.

The ssoadm utility has a set of subcommands that allow you to create and configure agents. All agent-related configurations that can be made using the OpenSSO Enterprise Console can also be made using the command line. This appendix indicates which ssoadm subcommands are related to agents.

An ssoadm Command-Line Example Specific to Agents

This section provides an example of how you can use the ssoadm command-line for agent-related subcommands. This example highlights the update-agent option. The update-agent option allows you to configure agent properties. The following is an example of how the ssoadm command can be issued with the update-agent option.

```
# ./ssoadm update-agent -e testRealm1 -b testAgent1 -u amadmin -f
/tmp/testpwd -a "com.sun.identity.agents.config.notenforced.url[0]=/exampleDir/public/*"
```

Remark D-1 **Reviewer** Please look at the command above. Are all the details of it accurate. It seems that the above example is different for web agents and J2EE agents. Besides that example above, does this appendix apply equally to web agents and J2EE agents? Are there any differences that should be addressed?

For the preceding command example, notice that a wildcard was used in the value for this particular property and that the property and value are enclosed in double quotes. The caution

that follows addresses this issue. For more information about wildcards, see [Appendix B, “Wildcard Matching in Policy Agent 3.0 Web Agents.”](#)



Caution – When issuing the `ssoadm` command, if you include values that contain wildcards (* or -*-), then the property name/value pair should be enclosed in double quotes to avoid substitution by the shell. This applies when you use the `-a (--attributevalues)` option. The double quotes are not necessary when you list the properties in a data file and access them with the `-D` option.

The format used to assign values to agent properties differs for the OpenSSO Enterprise Console and the `ssoadm` command-line utility. For the `ssoadm` utility, refer to the agent property files: `OpenSSOAgentBootstrap.properties` and `OpenSSOAgentConfiguration.properties`. These files demonstrate the correct format to use when assigning values to the agent properties using the `ssoadm` utility. Find these property files on the agent host machine in the following directory:

PolicyAgent-base/AgentInstance-Dir/config

Listing the Options for an ssoadm Subcommand

You can read the options for a subcommand from this guide or you can list the options yourself while using the command. On the machine hosting OpenSSO Enterprise, in the directory containing the `ssoadm` utility, issue the `ssoadm` command with the appropriate subcommand. For example:

```
# ./ssoadm update-agent
```

Since the preceding command is missing required options, the utility merely lists all the options available for this subcommand. For example:

```
ssoadm update-agent --options [--global-options]
Update agent configuration.
Usage:
ssoadm
    --realm|-e
    --agentname|-b
    --adminid|-u
    --password-file|-f
    [--set|-s]
    [--attributevalues|-a]
    [--datafile|-D]Global Options:
    --locale, -l
        Name of the locale to display the results.
```

```
--debug, -d
    Run in debug mode. Results sent to the debug file.

--verbose, -v
    Run in verbose mode. Results sent to standard output.
```

Options:

```
--realm, -e
    Name of realm.

--agentname, -b
    Name of agent.

--adminid, -u
    Administrator ID of running the command.

--password-file, -f
    File name that contains password of administrator.

--set, -s
    Set this flag to overwrite properties values.

--attributevalues, -a
    properties e.g. homeaddress=here.

--datafile, -D
    Name of file that contains properties.
```

Analysis of an ssoadm Subcommand's Usage Information

By looking at the usage information of a subcommand, you can determine which options are required and which are optional. You can list an option for the command with either a single letter, such as `-e` or with an entire word, such as `--realm`. The following is a list of the usage information for the `update-agent` subcommand:

```
ssoadm update-agent
  --realm|-e
  --agentname|-b
  --adminid|-u
  --password-file|-f
  [--set|-s]
  [--attributevalues|-a]
  [--datafile|-D]
```

The options not bounded by square brackets are required. Therefore, `realm`, `agentname`, `adminid`, `password-file`. However, even though the three options in brackets (the global options) are considered optional, you must use either `--attributevalues` or `--datafile` to

provide a property name and the corresponding value. The `--attributevalues` option is appropriate for assigning values to a single property. The `--datafile` option is appropriate for setting several properties at once. The `realm` and `agentname` options identify the specific agent you are configuring. The `adminid` and `password-file` commands identify you as someone who has the right to configure this agent.

The following command serves as an example of how you can change several agent properties at once. In this scenario the properties and their respective values are stored in a file, `/tmp/testproperties`, to which the command points:

```
# ./ssoadm update-agent -e testRealm1 -b testAgent1 -u amadmin -f
/tmp/testpwd -D /tmp/testproperties
```

Agent-Related Subcommands for the ssoadm Command

This sections lists the options available for each of the agent-related subcommands of the `ssoadm` command. The agent-related subcommands are presented as links in the following list:

- [“The ssoadm Command: add-agent-to-grp subcommand” on page 104](#)
- [“The ssoadm Command: agent-remove-props subcommand” on page 105](#)
- [“The ssoadm Command: create-agent subcommand” on page 106](#)
- [“The ssoadm Command: create-agent-grp subcommand” on page 107](#)
- [“The ssoadm Command: delete-agent-grps subcommand” on page 108](#)
- [“The ssoadm Command: delete-agents subcommand” on page 109](#)
- [“The ssoadm Command: list-agent-grp-members subcommand” on page 110](#)
- [“The ssoadm Command: list-agent-grps subcommand” on page 111](#)
- [“The ssoadm Command: list-agents subcommand” on page 111](#)
- [“The ssoadm Command: remove-agent-from-grp subcommand” on page 112](#)
- [“The ssoadm Command: show-agent subcommand” on page 113](#)
- [“The ssoadm Command: show-agent-grp subcommand” on page 114](#)
- [“The ssoadm Command: show-agent-membership subcommand” on page 115](#)
- [“The ssoadm Command: show-agent-types subcommand” on page 116](#)
- [“The ssoadm Command: update-agent subcommand” on page 116](#)
- [“The ssoadm Command: update-agent-grp subcommand” on page 117](#)

The ssoadm Command: add-agent-to-grp subcommand

```
ssoadm add-agent-to-grp --options [--global-options]
```

Add agents to a agent group.

Usage:

ssoadm


```
--realm|-e
--agentgroupname|-b
--agentnames|-s
--adminid|-u
--password-file|-f
```

Global Options:

```
--locale, -l
    Name of the locale to display the results.

--debug, -d
    Run in debug mode. Results sent to the debug file.

--verbose, -v
    Run in verbose mode. Results sent to standard output.
```

Options:

```
--realm, -e
    Name of realm.

--agentgroupname, -b
    Name of agent group.

--agentnames, -s
    Names of agents.

--adminid, -u
    Administrator ID of running the command.

--password-file, -f
    File name that contains password of administrator.
```

The ssoadm Command: agent-remove-props subcommand

```
ssoadm agent-remove-props --options [--global-options]
Remove agent's properties.
```

Usage:

```
ssoadm
    --realm|-e
    --agentname|-b
    --attributenames|-a
    --adminid|-u
    --password-file|-f
```

Global Options:

```
--locale, -l
    Name of the locale to display the results.

--debug, -d
    Run in debug mode. Results sent to the debug file.

--verbose, -v
    Run in verbose mode. Results sent to standard output.
```

Options:

```
--realm, -e
    Name of realm.

--agentname, -b
    Name of agent.

--attributenames, -a
    properties name(s).

--adminid, -u
    Administrator ID of running the command.

--password-file, -f
    File name that contains password of administrator.
```

The ssoadm Command: create-agent subcommand

```
ssoadm create-agent --options [--global-options]
```

Create a new agent configuration.

Usage:

ssoadm

```
--realm|-e
--agentname|-b
--agenttype|-t
--adminid|-u
--password-file|-f
[--attributevalues|-a]
[--datafile|-D]
```

Global Options:

```
--locale, -l
    Name of the locale to display the results.

--debug, -d
    Run in debug mode. Results sent to the debug file.
```

--verbose, -v
Run in verbose mode. Results sent to standard output.

Options:

--realm, -e
Name of realm.

--agentname, -b
Name of agent.

--agenttype, -t
Type of agent. e.g. J2EEAgent, WebAgent

--adminid, -u
Administrator ID of running the command.

--password-file, -f
File name that contains password of administrator.

--attributevalues, -a
properties e.g. homeaddress=here.

--datafile, -D
Name of file that contains properties.

The ssoadm Command: create-agent-grp subcommand

ssoadm create-agent-grp --options [--global-options]
Create a new agent group.

Usage:

ssoadm

--realm|-e
--agentgroupname|-b
--agenttype|-t
--adminid|-u
--password-file|-f
[--attributevalues|-a]
[--datafile|-D]

Global Options:

--locale, -l
Name of the locale to display the results.

--debug, -d

Run in debug mode. Results sent to the debug file.

--verbose, -v

Run in verbose mode. Results sent to standard output.

Options:

--realm, -e

Name of realm.

--agentgroupname, -b

Name of agent group.

--agenttype, -t

Type of agent group. e.g. J2EEAgent, WebAgent

--adminid, -u

Administrator ID of running the command.

--password-file, -f

File name that contains password of administrator.

--attributevalues, -a

properties e.g. homeaddress=here.

--datafile, -D

Name of file that contains properties.

The ssoadm Command: delete-agent-grps subcommand

ssoadm delete-agent-grps --options [--global-options]

Delete agent groups.

Usage:

ssoadm

--realm|-e

--agentgroupnames|-s

--adminid|-u

--password-file|-f

Global Options:

--locale, -l

Name of the locale to display the results.

--debug, -d

Run in debug mode. Results sent to the debug file.

```
--verbose, -v  
    Run in verbose mode. Results sent to standard output.
```

Options:

```
--realm, -e  
    Name of realm.  
  
--agentgroupnames, -s  
    Names of agent group.  
  
--adminid, -u  
    Administrator ID of running the command.  
  
--password-file, -f  
    File name that contains password of administrator.
```

The ssoadm Command: delete-agents subcommand

```
ssoadm delete-agents --options [--global-options]  
Delete agent configurations.
```

Usage:

```
ssoadm
```

```
--realm|-e  
--agentnames|-s  
--adminid|-u  
--password-file|-f
```

Global Options:

```
--locale, -l  
    Name of the locale to display the results.  
  
--debug, -d  
    Run in debug mode. Results sent to the debug file.  
  
--verbose, -v  
    Run in verbose mode. Results sent to standard output.
```

Options:

```
--realm, -e  
    Name of realm.  
  
--agentnames, -s  
    Names of agent.  
  
--adminid, -u
```

Administrator ID of running the command.

`--password-file, -f`

File name that contains password of administrator.

The ssoadm Command: list-agent-grp-members subcommand

`ssoadm list-agent-grp-members --options [--global-options]`

List agents in agent group.

Usage:

`ssoadm`

`--realm|-e`

`--agentgroupname|-b`

`--adminid|-u`

`--password-file|-f`

`[--filter|-x]`

Global Options:

`--locale, -l`

Name of the locale to display the results.

`--debug, -d`

Run in debug mode. Results sent to the debug file.

`--verbose, -v`

Run in verbose mode. Results sent to standard output.

Options:

`--realm, -e`

Name of realm.

`--agentgroupname, -b`

Name of agent group.

`--adminid, -u`

Administrator ID of running the command.

`--password-file, -f`

File name that contains password of administrator.

`--filter, -x`

Filter (Pattern).

The ssoadm Command: list-agent-grps subcommand

ssoadm list-agent-grps --options [--global-options]
List agent groups.

Usage:

```
ssoadm
  --realm|-e
  --adminid|-u
  --password-file|-f
  [--filter|-x]
  [--agenttype|-t]
```

Global Options:

```
--locale, -l
    Name of the locale to display the results.

--debug, -d
    Run in debug mode. Results sent to the debug file.

--verbose, -v
    Run in verbose mode. Results sent to standard output.
```

Options:

```
--realm, -e
    Name of realm.

--adminid, -u
    Administrator ID of running the command.

--password-file, -f
    File name that contains password of administrator.

--filter, -x
    Filter (Pattern).

--agenttype, -t
    Type of agent. e.g. J2EEAgent, WebAgent
```

The ssoadm Command: list-agents subcommand

ssoadm list-agents --options [--global-options]
List agent configurations.

Usage:

```
ssoadm
  --realm|-e
```

```
--adminid|-u
--password-file|-f
[--filter|-x]
[--agenttype|-t]
```

Global Options:

```
--locale, -l
    Name of the locale to display the results.

--debug, -d
    Run in debug mode. Results sent to the debug file.

--verbose, -v
    Run in verbose mode. Results sent to standard output.
```

Options:

```
--realm, -e
    Name of realm.

--adminid, -u
    Administrator ID of running the command.

--password-file, -f
    File name that contains password of administrator.

--filter, -x
    Filter (Pattern).

--agenttype, -t
    Type of agent. e.g. J2EEAgent, WebAgent
```

The ssoadm Command: remove-agent-from-grp subcommand

ssoadm remove-agent-from-grp --options [--global-options]
Remove agents from a agent group.

Usage:

```
ssoadm --realm|-e
    --agentgroupname|-b
    --agentnames|-s
    --adminid|-u
    --password-file|-f
```

Global Options:

```
--locale, -l
```


Name of the locale to display the results.

--debug, -d

Run in debug mode. Results sent to the debug file.

--verbose, -v

Run in verbose mode. Results sent to standard output.

Options:

--realm, -e

Name of realm.

--agentgroupname, -b

Name of agent group.

--agentnames, -s

Names of agents.

--adminid, -u

Administrator ID of running the command.

--password-file, -f

File name that contains password of administrator.

The ssoadm Command: show-agent subcommand

ssoadm show-agent --options [--global-options]

Show agent profile.

Usage:

ssoadm

--realm|-e

--agentname|-b

--adminid|-u

--password-file|-f

[--outfile|-o]

[--inherit|-i]

Global Options:

--locale, -l

Name of the locale to display the results.

--debug, -d

Run in debug mode. Results sent to the debug file.

--verbose, -v

Run in verbose mode. Results sent to standard output.

Options:

- `--realm, -e`
Name of realm.
- `--agentname, -b`
Name of agent.
- `--adminid, -u`
Administrator ID of running the command.
- `--password-file, -f`
File name that contains password of administrator.
- `--outfile, -o`
Filename where configuration is written to.
- `--inherit, -i`
Set this to inherit properties from parent group.

The ssoadm Command: show-agent-grp subcommand

`ssoadm show-agent-grp --options [--global-options]`
Show agent group profile.

Usage:

`ssoadm`

- `--realm|-e`
- `--agentgroupname|-b`
- `--adminid|-u`
- `--password-file|-f`
- `[--outfile|-o]`

Global Options:

- `--locale, -l`
Name of the locale to display the results.
- `--debug, -d`
Run in debug mode. Results sent to the debug file.
- `--verbose, -v`
Run in verbose mode. Results sent to standard output.

Options:

- `--realm, -e`
Name of realm.

```
--agentgroupname, -b
    Name of agent group.

--adminid, -u
    Administrator ID of running the command.

--password-file, -f
    File name that contains password of administrator.

--outfile, -o
    Filename where configuration is written to.
```

The ssoadm Command: show-agent-membership subcommand

```
ssoadm show-agent-membership --options [--global-options]
List agent?s membership.
```

Usage:

```
ssoadm
    --realm|-e
    --agentname|-b
    --adminid|-u
    --password-file|-f
```

Global Options:

```
--locale, -l
    Name of the locale to display the results.

--debug, -d
    Run in debug mode. Results sent to the debug file.

--verbose, -v
    Run in verbose mode. Results sent to standard output.
```

Options:

```
--realm, -e
    Name of realm.

--agentname, -b
    Name of agent.

--adminid, -u
    Administrator ID of running the command.

--password-file, -f
    File name that contains password of administrator.
```

The ssoadm Command: show-agent-types subcommand

ssoadm show-agent-types --options [--global-options]

Show agent types.

Usage:

ssoadm

--adminid|-u
--password-file|-f

Global Options:

--locale, -l
Name of the locale to display the results.

--debug, -d
Run in debug mode. Results sent to the debug file.

--verbose, -v
Run in verbose mode. Results sent to standard output.

Options:

--adminid, -u
Administrator ID of running the command.

--password-file, -f
File name that contains password of administrator.

The ssoadm Command: update-agent subcommand

ssoadm update-agent --options [--global-options]

Update agent configuration.

Usage:

ssoadm

--realm|-e
--agentname|-b
--adminid|-u
--password-file|-f
[--set|-s]
[--attributevalues|-a]
[--datafile|-D]

Global Options:

--locale, -l
Name of the locale to display the results.

--debug, -d
Run in debug mode. Results sent to the debug file.

--verbose, -v
Run in verbose mode. Results sent to standard output.

Options:

--realm, -e
Name of realm.

--agentname, -b
Name of agent.

--adminid, -u
Administrator ID of running the command.

--password-file, -f
File name that contains password of administrator.

--set, -s
Set this flag to overwrite properties values.

--attributevalues, -a
properties e.g. homeaddress=here.

--datafile, -D
Name of file that contains properties.

The ssoadm Command: update-agent-grp subcommand

ssoadm update-agent-grp --options [--global-options]

Update agent group configuration.

Usage:

```
ssoadm
  --realm|-e
  --agentgroupname|-b
  --adminid|-u
  --password-file|-f
  [--set|-s]
  [--attributevalues|-a]
  [--datafile|-D]
```

Global Options:

--locale, -l

Name of the locale to display the results.

--debug, -d

Run in debug mode. Results sent to the debug file.

--verbose, -v

Run in verbose mode. Results sent to standard output.

Options:

--realm, -e

Name of realm.

--agentgroupname, -b

Name of agent group.

--adminid, -u

Administrator ID of running the command.

--password-file, -f

File name that contains password of administrator.

--set, -s

Set this flag to overwrite properties values.

--attributevalues, -a

properties e.g. homeaddress=here.

--datafile, -D

Name of file that contains properties.

Web Agent Error Codes

This appendix lists the error codes you might encounter while installing and configuring a web agent. It also provides explanations for the each code item.

Error Code List

This list of error codes includes locations that are reserved for error codes that do not currently exist.

0. AM_SUCCESS	The operation completed successfully.
1. AM_FAILURE	The operation did not complete successfully. Please refer to the log file for more details.
2. AM_INIT_FAILURE	The C SDK initialization routine did not complete successfully. All the other APIs may be used only if the initialization went through successfully.
3. AM_AUTH_FAILURE	The authentication did not go through successfully. This error is returned either by the Authentication API or the Policy Initialization API, which tries to authenticate itself as a client to OpenSSO Enterprise.
4. AM_NAMING_FAILURE	The naming query failed. Please look at the log file for further information.
5. AM_SESSION_FAILURE	The session operation did not succeed. The operation may be any of the operations provided by the session API.
6. AM_POLICY_FAILURE	The policy operation failed. Details of policy failure may be found in the log file.

7. This is a reserved error code.	Currently, no error code exists at this location.
8. AM_INVALID_ARGUMENT	The API was invoked with one or more invalid parameters. Check the input provided to the function.
9. This is a reserved error code.	Currently, no error code exists at this location.
10. This is a reserved error code.	Currently, no error code exists at this location.
11. AM_NO_MEMORY	The operation failed because of a memory allocation problem.
12. AM_NSPR_ERROR	The underlying NSPR layer failed. Please check log for further details.
13. This is a reserved error code.	Currently, no error code exists at this location.
14. AM_BUFFER_TOO_SMALL	The web agent does not have memory allocated to receive data from OpenSSO Enterprise.
15. AM_NO_SUCH_SERVICE_TYPE	The service type input by the user does not exist. This is a more specific version of AM_INVALID_ARGUMENT error code. The error can occur in any of the API that take <code>am_policy_t</code> as a parameter.
16. AM_SERVICE_NOT_AVAILABLE	Currently, no error code exists at this location.
17. AM_ERROR_PARSING_XML	During communication with OpenSSO Enterprise, there was an error while parsing the incoming XML data.
18. AM_INVALID_SESSION	The session token provided to the API was invalid. The session may have timed out or the token is corrupted.
19. AM_INVALID_ACTION_TYPE	This exception occurs during policy evaluation, if such an action type does not exist for a given policy decision appropriately found for the resource.
20. AM_ACCESS_DENIED	The user is denied access to the resource for the kind of action requested.
21. AM_HTTP_ERROR	There was an HTTP protocol error while contacting OpenSSO Enterprise.
22. AM_INVALID_FQDN_ACCESS	The resource provided by the user is not a fully qualified domain name. This is a web container

	specific error and may be returned by the <code>am_web_is_access_allowed</code> function only.
23. AM_FEATURE_UNSUPPORTED	The feature being invoked is not implemented as of now. Only the interfaces have been defined.
24. AM_AUTH_CTX_INIT_FAILURE	The Auth context creation failed. This error is thrown by <code>am_auth_create_auth_context</code> .
25. AM_SERVICE_NOT_INITIALIZED	The service is not initialized. This error is thrown by <code>am_policy</code> functions if the provided service was not initialized previously using <code>am_policy_service_init</code> .
26. AM_INVALID_RESOURCE_FORMAT	This is a plug-in interface error. Implementors of the new resource format may throw this error if the input string does not meet their specified format. This error is thrown by the <code>am_web</code> layer, if the resource passed as parameter does not follow the standard URL format.
27. AM_NOTIF_NOT_ENABLED	This error is thrown if the notification registration API is invoked when the notification feature is disabled in the configuration file.
28. AM_ERROR_DISPATCH_LISTENER	Error during notification registration.
29. AM_REMOTE_LOG_FAILURE	This error code indicates that the service that logs messages to OpenSSO Enterprise has failed. The details of this error can be found in the web agent's log file.

Developing Your Own OpenSSO Enterprise Web Agent

You can develop web agents yourself for OpenSSO Enterprise Policy Agent. By using the available C application programming interfaces (C API) and by following the provided samples, developing your own web agent can be a relatively simple development task, depending on the web container.

A web agent that suits your site's requirements might already exist in the OpenSSO Enterprise Policy Agent software set. If the web agent exists, you can download it from the agents download page. However, if a web agent is not currently available for a particular web container or platform, you can develop that agent yourself using the web agent development toolkit.

The Web Agent Development Toolkit

The web agent development toolkit refers to tools available for developing your own web agent for OpenSSO Enterprise Policy Agent. The basic tools of the toolkit are as follows:

- The C SDK (C software development kit), which comes with the OpenSSO Enterprise download.

The C SDK includes the C API and sample source files that you can use as a blueprint for creating a web agent.

This kit is not only for developing agents, though that is the focus of this appendix. This kit can be used to enable external C applications to participate in OpenSSO Enterprise authentication, authorization, single sign-on (SSO), and logging operations. The C SDK is available as a .zip file in the following directory:

`zip-root/opensso/libraries/native/agent-csdk`

See the README.TXT file in the samples directory of the C SDK for more information about using this web agent development toolkit.

- *Sun OpenSSO Enterprise 8.0 C API Reference for Application and Web Policy Agent Developers*

This guide describes the C API, providing information to help you create your own web agent.

By using the C SDK and the C API Reference described in the preceding list, you can create your own web agent that is compatible with OpenSSO Enterprise.

Index

A

- advice, composite, 80
- agent cache, updating, 63-64
- agent profile password
 - encryption, 74-76
 - resetting
 - Linux systems, 76
 - Solaris systems, 75
 - Windows systems, 76
- agentadmin command, 41-49
 - agentInfo, 46-47
 - help, 49
 - install, 42-44
 - listAgents, 45-46
 - uninstall, 44-45
 - uninstallAll, 47-48
 - usage, 48-49
 - version, 47
- agentadmin program, 41-49
- authentication, 39-40
 - level, 23, 39
 - definition of, 40
 - module
 - definition of, 40
 - examples of, 40
 - specified protection for, 66
 - user, 22, 27, 33, 35

B

- backup deployment container, 63

C

- cache, updating, 63-64
- cascading style sheets (CSS)
 - not-enforced list
 - URL, 65
- CDSSO, configuring, 72-73
- client IP addresses, validating, 74
- compatibility, OpenSSO Enterprise, J2EE agents, 17
- composite advice, 80
- configuring, CDSSO, 72-73
- cookies, resetting, 72
- cross-domain single sign-on, 20
- cross domain single sign-on, configuring, 72-73

E

- enabling, load balancing, 78-80
- encryption, agent profile password, 74-76
- error codes, 119-121
- expiration mechanism, cache, 63-64

F

- failover protection, 63
- FQDN
 - mapping
 - turning off, 71
 - setting, 70

fully qualified domain name
 mapping
 turning off, 71
 setting, 70

G

generating
 state file
 installation, 81-82
.gif image
 not-enforced list
 URL, 65

H

high availability, 63
hijacking
 single sign-on (SSO)
 tokens, 74
hybrid agent cache, updating, 64

I

installation
 silent, 81-84
 using state file, 82-83
inverted
 not-enforced list
 URL, 65

J

J2EE
 security, 35, 38

L

load balancing, enabling, 78-80
logging, access attempt, 20

N

not-enforced list
 IP address, 66
 URL, 65
 inverted, 65
notification mechanism, cache, 63-64

O

OpenSSO Enterprise
 Service
 Authentication, 20, 22, 27
 service
 definition of, 40
 Service
 Logging, 20
 Naming, 20
 Policy, 20, 28
 Session, 20, 35

P

personalization
 policy-based response attributes, 68-69
 session attributes, 67-68
 user profile attributes, 69-70
policy, definition of, 40
policy-based
 response attributes
 personalization, 68-69
policy decision
 enforcement, 22, 37
 process, 23
portal server, 34, 37

R

redirect
 browser, 27, 35
REMOTE_USER variable, setting, 73

- resetting
 - agent profile password
 - Linux systems, 76
 - Solaris systems, 75
 - Windows systems, 76
 - cookies, 72
- response
 - attributes
 - mapping, 69

S

- security
 - J2EE, 35, 38
- service
 - definition of, 40
 - OpenSSO Enterprise, 20
- session
 - attributes
 - personalization, 67-68
 - cache
 - updating, 63-64
 - token, 28
 - web server agent, 34
- silent
 - installation, 81-84
 - uninstallation, 81-84
- single sign-on, enforcement, 37
- state file
 - for installation, 82-83
 - for uninstallation, 84
 - generating, 81-82
 - uninstallation, 83-84

U

- uninstallation
 - silent, 81-84
 - using state file, 83-84, 84
- updating, agent cache, 63-64
- URI, 37
- user
 - authentication, 22, 27, 33, 35

- user authentication, 39-40
- user profile, attributes, 69-70

W

- web agent, error codes, 119-121
- web server, embedded, 38

