CA SiteMinder®

Web Agent Configuration Guide

r12.0 SP2

CA Product References

This document references the following CA products:

- CA SiteMinder®
- CA Wily Introscope®
- CA CA Identity Manager
- CA SOA Security Manager

Contact CA

Contact Technical Support

For your convenience, CA provides one site where you can access the information you need for your Home Office, Small Business, and Enterprise CA products. At http://ca.com/support, you can access the following:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Provide Feedback

If you have comments or questions about CA product documentation, you can send a message to techpubs@ca.com.

If you would like to provide feedback about CA product documentation, complete our short customer survey, which is also available on the CA Support website, found at http://ca.com/docs.

Contents

Chapter 1: Web Agents

This section contains the following topics:

How Web Agents Secure Resources

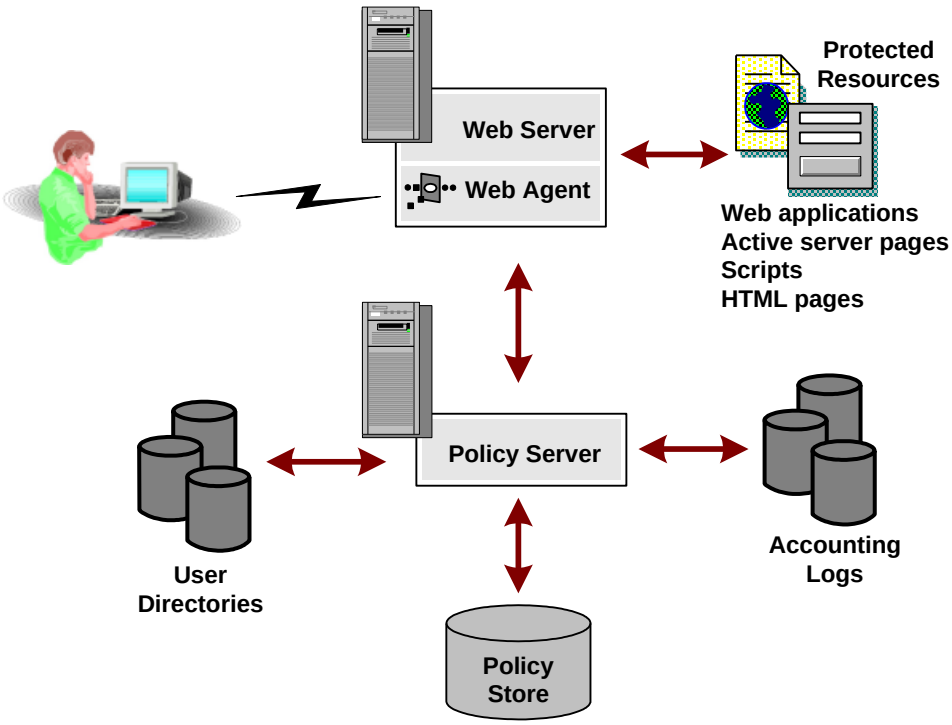
A SiteMinder Web Agent is a software component that controls access to any resource that can be identified by a URL. The Web Agent resides on a web server and intercepts requests for a resource to determine whether or not the resource is protected by SiteMinder. The Web Agent then interacts with the Policy Server to authenticate and authorize users who request access to the protected web server resources.

Web Agents perform the following tasks:

- Intercept access requests for protected resources and work with the Policy Server to determine whether or not a user should have access.
- Provide information to a Web application that dictates how content is presented to the user (policy-based personalization) and how to deliver access privileges.
- Ensure a user's ability to access information quickly and securely. Web Agents store contextual information about user access privileges in a session cache. You can optimize performance by modifying the cache settings.
- Enable single sign-on across Web servers in a single cookie domain or across multiple cookie domains without requiring users to re-authenticate.

For a list of SiteMinder Web Agents and supported Web server platforms, go to Technical Support and search for SiteMinder Support Matrix.

Web Agents reside on web servers as illustrated in the following diagram:


Web Server
Web Agent
Protected Resources
Web applications
Active server pages
Scripts
HTML pages
Policy Server
User Directories
Accounting Logs
Policy Store

How Web Agents and the Policy Server Work Together

To enforce access control, the Web Agent interacts with the Policy Server, where all authentication and authorization decisions are made.

The Web Agent intercepts user requests for resources and checks with the Policy Server to see if the requested resource is protected. If the resource is unprotected, the access request proceeds directly to the web server. If the resource is protected, the following occurs:

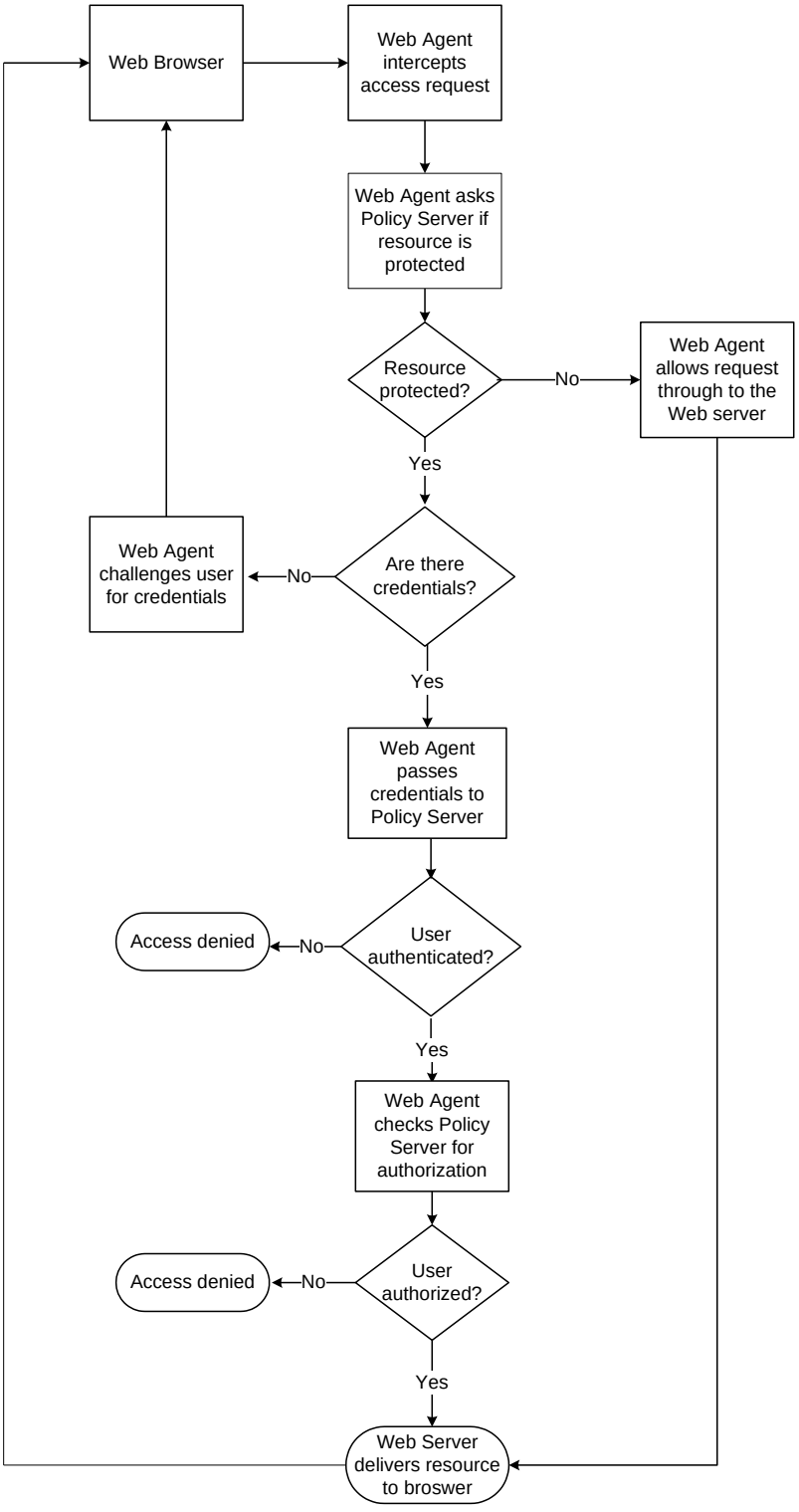
1. The Web Agent checks which authentication method is required for this resource. Typical credentials are a name and password, but other credentials, such as a certificate or a token card PIN, may be required.
2. The Web Agent challenges the user for credentials.

   The user responds with the appropriate credentials.
3. The Web Agent passes the credentials to the Policy Server, which determines if the credentials are correct.
4. If the user passes the authentication phase, the Policy Server determines if the user is authorized to access the resource. Once the Policy Server grants access, the Web Agent allows the request to proceed to the web server.

The Web Agent also receives user-specific attributes, in the form of a *response,* to enable Web content personalization and session management. A response is a personalized message or other user-specific information returned to the Web Agent from the Policy Server after authorizing the user. It consists of name-value attribute pairs that are added to HTTP headers by the Web Agent for use with Web applications. Examples of responses include the following:

- After authorizing a user to access a Web application, the Web Agent could also send information to the Web application dictating how long the user session can last.
- If the user is returning to a site where he or she previously registered, the Web Agent could return information about that user's buying preferences.

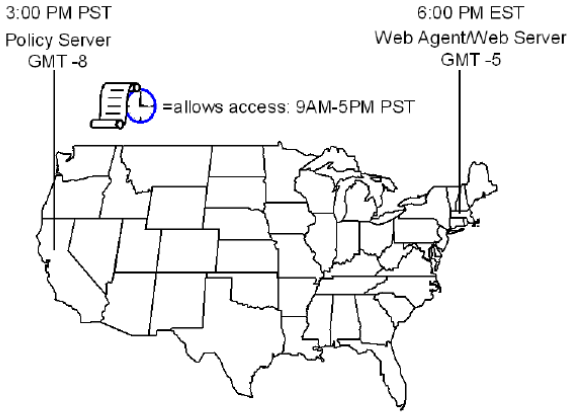
The following diagram shows the communication between the Web Agent and the Policy Server:


Web Browser
Web Agent intercepts access request
Web Agent asks Policy Server if resource is protected
Resource protected?
No
Web Agent allows request through to the Web server
Yes
Are there credentials?
No
Web Agent challenges user for credentials
Yes
Web Agent passes credentials to Policy Server
User authenticated?
No
Access denied
Yes
Web Agent checks Policy Server for authorization
User authorized?
No
Access denied
Yes
Web Server delivers resource to broswer

Considerations for Web Agents and Policy Servers in Different Time Zones

By default, the Policy Server and Web Agent calculate time relative to Greenwich Mean Time (GMT). Therefore, for each system that has a Policy Server or Web Agent installed, the system clock must be set for the time zone appropriate to that system's geographical location.

The following illustration shows how the Policy Server executes a policy relative to time. A resource is stored on a web server in Massachusetts and is protected by a Policy Server in California. The policy allows access to the resource between 9:00 AM and 5:00 PM. However, the user in Massachusetts can still access the resource at 6:00 PM because the policy is based on the Policy Server's time zone, Pacific Standard Time (PST), which is three hours behind the Web Agent's time zone, Eastern Standard Time (EST).



At 6:00 PM, the user in Massachusetts can still access the resource because it is only 3:00PM according to the Policy Server.

Note: For Windows systems, both the time zone setting and the time of day (set in the Date/Time control panel) must agree. For example, to reset a system in the U.S. from Eastern time to Pacific time, perform the following tasks in the order shown:

a. Set the time zone to Pacific time.

b. Verify that the system clock displays the correct time (three hours earlier than Eastern time).

If these settings differ, single sign-on across multiple domains and agent key management will not work properly.

How the Agent Reads SiteMinder Cookies

Web Agents use agent keys to encrypt and decrypt SiteMinder cookies so the data they contain can be read. The Agent uses the key to encrypt cookies before sending them to a user's browser and to decrypt cookies received from other Web Agents.

All Web Agents need to be aware of the same keys, and the keys must be set to the same value for all Agents communicating with a Policy Server. This rule is particularly important for Agents in a single sign-on environment. To ensure that the keys remain secure, the Policy Server performs a key *rollover*. A key rollover is the process of generating new keys, encrypting them, and distributing them to all Web Agents within a SiteMinder environment.

When a Web Agent starts up and makes a management call request, the Policy Server supplies the current set of keys. Each time the Web Agent polls the Policy Server, the agent again makes the management call. The Web Agent receives the updated keys.

The Policy Server provides the following types of keys:

Dynamic Keys

Refers to a key that is generated by a Policy Server algorithm and distributed to other connected Policy Servers and their associated Web Agents. Dynamic keys can be rolled over automatically at a regular interval, or they can be changed manually by using the Administrative UI.

Static Keys

Refers to a key that remains the same indefinitely, and can be generated by a Policy Server algorithm or configured manually. SiteMinder uses this type of key for a subset of features that requires information to be stored in cookies over extended periods.

Automated key changes ease the process of managing agent keys for large SiteMinder installations that share a single *key store*. A key store is a storage location for all key information. Policy Servers access the key store to obtain the current keys, which are then passed on to the Web Agents. For Agents that are configured for single sign-on, the key store must be replicated and shared across all Policy Servers in the single sign-on environment. Automating key changes also ensures the integrity of the keys.

Note: For more information, see the Policy Server documentation.

Web Agents and Dynamic Key Rollovers

You can use the Administrative UI to configure dynamic Agent key rollover. Web Agents poll the Policy Server for key updates at regular intervals. If keys have been updated, Web Agents pick up the changes during polling. The default polling time is 30 seconds, but you can customize it by changing the value of the PSPollInterval parameter for a Web Agent.

When a Web Agent detects that a key rollover has occurred, the Agent retrieves new values for the following Agent keys:

Old Key

Contains the last value used for the dynamic Agent key before the current value.

Current Key

Contains the value of the current dynamic Agent key.

Future Key

Contains the next value that will be used as the *current key* in a dynamic Agent key rollover.

Static Key

Contains a long-term key that the Agent can use for SiteMinder features that need to identify a user and maintain this information for long periods. Static keys also support cookie encryption for single sign-on when dynamic keys are not enabled.

Web Agents require multiple keys to preserve cookie data and ensure a smooth transition between old keys and new keys.

More information:

Change How Often an Agent Checks for Policy or Key Updates (see page 52)

Key Stores

When the Policy Server generates dynamic keys, it saves and maintains these keys in the key store. The key store is a repository from which all Policy Servers retrieve the most current keys. Web Agents obtain the current keys from the Policy Servers. The key store may be part of a SiteMinder policy store or maintained as a stand-alone key store.

Note: If an administrator issues multiple agent key rollovers in rapid succession, this action may invalidate all cookies issued for single sign-on and may disrupt single sign-on for all users currently logged in. After these users re-authenticate, single sign-on will operate normally.

Types of Web Agents (Traditional and Framework)

All SiteMinder Web Agents are based on one of two architectures. Traditional Web Agents are based on the original SiteMinder Agent architecture. Framework Agents were introduced with SiteMinder version 5.x QMR 6. Features provided by the Web Agent are primarily the same regardless of the architecture. There are some differences. Framework Agents, for example, use a different WebAgent.conf file and a LocalConfig.conf file, that are not used by the traditional Web Agents.

Traditional Web Agents are installed on the following web servers:

- Apache 1.x
- Apache 1.x-based servers, such as the IBM HTTP Server
- Domino

Framework Agents are installed on the following web servers:

- IIS 6.0
- Apache 2.0
- Apache 2.0-based servers: IBM HTTP Server, HP Apache server
- Sun Java Systems 6.0 and 6.1

  Note: The Sun Java System Web server was formerly called the Sun ONE Web server or the iPlanet Web server.

More information:

How to Meet the Use Case Deployment Prerequisites (see page 41)
Enable Post Preservation between Framework and Traditional Agents (see page 245)

Parameters Not Used by Framework Agents

The following parameters are used by traditional agents only and are not used in Framework agents:

DisablePostDataLimit

Specifies whether a Web Agent observes the 64 KB data-size limit when preserving or filtering POST data. This does not affect the standard POST operation, but it does affect the following:

- POST preservation
- eTelligent Rules Post variables
- TransactionMinder authentication schemes

Default: No (limit enforced)

Important! Change this parameter to yes at your own risk.

Note: This parameter applies to IIS 5.0 Web Agents only.

InsecureServer

Allows Web Agents for IIS 5 to enforce native IIS security mechanisms by providing a Windows user security context. Add this parameter to the Agent Configuration Object or local configuration file with the value you want.

If this parameter is set to yes, the Web Agent stores encrypted credentials in paged memory, which can be written to the operating system's page file and saved to a hard disk.

Important! If your hard disk is stolen or compromised, confidential data could be exposed.

If this parameter is no, the Web Agent stores encrypted credentials in protected kernel memory. This setting is more secure, but it places more demands on the physical memory of your IIS server.

Default: No

Parameters Requiring a Server Restart when Changed

Some Agent parameters be updated dynamically. You must restart the web server to apply any changes to the following parameters:

AgentConfigObject

Defines the name of an Agent Configuration Object (stored on a policy server) in a local agent configuration file. This parameter is *not* used in Agent Configuration Objects.

Default: no default

CacheAnonymous

Specifies if the Web Agent caches anonymous user information. You may want to set this parameter in any of the following situations:

- If your web site gets mostly anonymous users and you want to store their session information.
- If your web site gets a mix of registered and anonymous users.

  You may want to disable this parameter to keep the anonymous user information from filling the cache and leaving no room for registered users.

Default: No

HostConfigFile

Specifies the path to the SMHost.conf file (in an IIS 6.0 or Apache agent) that is created after a trusted host computer has been successfully registered with a Policy server. All Web Agents on a computer share the SMHost.conf file.

Default: No default

MaxResourceCacheSize

Specifies the maximum number of entries that the Web Agent keeps in its resource cache. An entry contains the following information:

- A Policy Server response about whether a resource is protected
- Any additional attributes returned with the response

When the maximum is reached, new resource records replace the oldest resource records.

If you set this value to a high number, be sure that sufficient system memory is available.

If you are viewing Web Agent statistics using the OneView Monitor, you may notice that the value shown for the ResourceCacheCount is greater than the value you specified for the MaxResourceCacheSize parameter. This is not an error. The Web Agent uses the MaxResourceCacheSize parameter as a guideline and the values may at times differ because the MaxResourceCacheSize parameter represents the maximum number of average-sized entries in the resource cache. The actual cache entries are most likely larger or smaller than the pre-determined average size; therefore, the effective maximum number of entries may be more or less than the value specified.

Note: For Web Agents that use shared memory, such as the Agent on Apache 1.x and the framework Agents, the cache is pre-allocated to a constant size based on the MaxResourceCacheSize value and will not grow.

Default: (Domino web servers) 1000

Default: (IIS and Sun Java System web servers) 700

Default: (Apache web servers) 750

MaxSessionCacheSize

Specifies the maximum number of users the Agent maintains in its session cache. The session cache stores the session IDs of users who authenticate successfully. If those users access another resource in the same realm during the same session, the Agent uses the information from the session cache instead of calling the Policy Server. When the maximum number is reached, the Agent replaces the oldest user records with new user records.

Base the value of this parameter on the number of users that you expect to access and use resources for a sustained period of time. If you set this value to a high number, ensure that sufficient system memory is available.

Default: (Domino web servers) 1000

Default: (IIS and Sun Java System web servers) 700

Default: (Apache web servers) 750

PostPreservationFile

Enables the transfer of POST preservation data between Traditional and Framework Agents by specifying the path to *one* of the following POST-preservation-template files:

- tr2fw.pptemplate—Indicates that resources hosted on a server running a traditional agent are protected by an FCC running on a Framework agent.
- fw2tr.pptemplate—Indicates that resources hosted on a server running a Framework agent are protected by an FCC running on a Traditional agent.

Default: No default

Example: *web_agent_home*/samples/forms/fw2tr.pptemplate

ResourceCacheTimeout

Specifies the number of seconds that resource entries remain in the cache. If a user tries to access a protected resource after the time interval has been exceeded, the Web Agent removes the cached entries and contacts the Policy server.

Default: 600 (10 minutes)

More information:

Implement Central Configuration (see page 29)

Web Agent Configurations

A Web Agent's configuration determines how it receives the values for its parameters. You can configure a Web Agent with one (or a combination of both) of the following methods:

Central

Receives parameter values from an Agent Configuration Object on the Policy Server.

Local

Receives parameter values from a file installed on the system hosting the web server.

The Web Agent reads its configuration settings using the following process:

1. When a Web Agent is enabled, it searches the Agent Configuration Object (on the Policy Server) for configuration information.
2. The Web Agent examines the value of the AllowLocalConfig parameter.
3. If AllowLocalConfig is set to no, the Web Agent retrieves all of its configuration settings from the Agent Configuration object. If AllowLocalConfig is set to yes, the Web Agent searches the corresponding agent's local configuration file for modified or additional parameters. The settings in the local configuration file override those in the Agent Configuration object.
4. The Web Agent creates a unified local copy of an Agent Configuration object using the settings from the central and local sources. This local copy does not change the original Agent Configuration object on the Policy Server.

More Information

Central and Local Configuration Together (see page 39)

Central Configuration

A central agent configuration manages one or more Web Agents from an Agent Configuration Object in the Policy Server. The Agent Configuration Object that resides in the Policy Server contains the parameters used by the Web Agents. One advantage of central configuration is that you can update the parameter settings of several agents at once. Most parameter changes occur dynamically, but some Framework parameters require a web server restart after they are changed.

You create and edit an Agent Configuration Object with the Administrative UI. Each Web Agent communicating with the Policy Server must be associated with an Agent Configuration Object, but many Web Agents can use a single Agent Configuration Object.

Note: For more information about creating an Agent Configuration Object, see the Policy Server documentation.

More Information

Parameters Requiring a Server Restart when Changed (see page 25)

Implement Central Configuration

Central configuration is enabled by default. The Web Agent uses the configuration settings from the existing Agent Configuration Object that you specified when you configured the Web Agent with the configuration wizard. You can change the settings of the parameters to suit your needs at any time.

To implement central configuration

1. Log into the Administrative UI.

   The Welcome screen appears.

2. Click Infrastructure, Agent Configuration.

   A list of agent configuration tasks appears.

3. Click Modify Agent Configuration.

   The search window appears.

4. (Optional) Fill out the search form to narrow your search criteria.

5. Click Search.

   A list of Agent Configuration objects appears.

6. Click the radio button to the left of the Agent Configuration Object associated with the Agent on which you want to implement central configuration. Click Select.

   The Modify Agent Configuration window appears.

7. Ensure the value of the AllowLocalConfig parameter is set to no.

8. Use the Administrative UI to add, edit or delete the settings of any other parameters according to your needs.

9. Click Submit when you are finished making changes.

   The Modify Agent Configuration window closes, and a confirmation message appears. Central configuration is implemented. Most parameter changes occur dynamically, but some changes require a web server restart to take effect.

More information:

Parameters Requiring a Server Restart when Changed (see page 25)

Local Agent Configuration

Local Configuration

A local agent configuration manages a Web Agent using local files that are installed on the system hosting the web server. The parameter settings in the local file override any settings stored in an Agent Configuration Object on the Policy Server. The settings in the Agent Configuration Object do not change. Situations to consider local agent configuration include the following:

- When you have three Apache Web Agents, and the first two (A and B) use identical parameter settings, but you want the third Apache Agent (C) use the most of the settings from A and B while acting as a reverse proxy. To accomplish this, use central agent configuration for Apache Agents A and B, but use local configuration for Apache Agent C.

- When the Policy Server administrator is not the same person (or group) who configures an Agent. For example, the information technology department in a company maintains the Policy Server, but the finance department uses an Agent to control access to an accounting application. Someone from the information technology department enables local configuration for the Agent on the Policy Server, but another person from the finance department controls the specific configuration settings for the Agent that protects the accounting application.

Framework Web Agents use the following files for local configuration:

WebAgent.conf

Contains the core settings that the Framework Web Agent uses to start and connect to a Policy Server.

LocalConfig.conf

Contains the configuration settings for the Framework Web Agents.

Traditional Web Agents use the following file for local configuration:

WebAgent.conf

Contains all of the configuration settings for traditional Web Agents.

WebAgent.conf File Locations

The following table shows where the WebAgent.conf file is created on various web servers:

Web Server	File Location
IIS	Program Files\ca\webagent\bin\IIS

Web Server	File Location
Sun Java System (iPlanet/SunOne)	*Sun_Java_server_home*/https-*hostname*/config where *Sun_Java_server_home* is the location in which the Sun Java System web server is installed and *hostname* is the name of the server.
Apache, IBM HTTP Server Oracle HTTP Server	*web_server_home*/conf where *web_server_home* is the installed location of the web server
Domino	Windows: c:\lotus\domino UNIX: $HOME/notesdata

WebAgent.conf file for Framework Agents

Along with the AgentConfigObject, HostConfigFile, and EnableWebAgent parameters, the following parameters are also added to the WebAgent.conf file of Framework Agents:

Important! You should not modify any sections of the file that refer to other SiteMinder products besides the Web Agent. You may, however, change the values of the Web Agent parameters in the file.

LocalConfigFile

Specifies the location of the LocalConfig.conf file, where most of Agent configuration settings reside.

ServerPath

Identifies the web server directory (of Apache 2.0 and Sun Java System web servers) to the Agent.

LoadPlugin

Specifies which plug-ins are loaded for IIS 6.0 and Apache 2.0 Agents. The plug-ins support different types of Agent functions. The following plug-ins are available:

HttpPlugin

Specifies whether the Web Agent operates as an HTTP agent.

Default: Enabled

SAMLAffiliatePlugin

Allows communication between the Web Agent and a SAML Affiliate Agent (if you have purchased Federation Security Services).

Default: Disabled

Affiliate10Plugin

Allows communication between the Web Agent and a 4.x Affiliate Agent. This is not used by the SAML Affiliate Agent.

Default: Disabled

To enable the other LoadPlugin entries, remove the pound symbol (#) from the beginning of the line.

More Information

Manage Web Agents with Multiple Web Server Instances (see page 54)

LocalConfig.conf File Locations (Framework Agents)

When you install a Framework Web Agent, the SiteMinder installation program creates a LocalConfig.conf file in the following directory:

Windows

web _agent_home\config

UNIX

web _agent_home/config

Important! This file contains all of the default settings. Do not modify this file. We recommend creating a backup copy of this file for future reference or for recovery purposes.

When you configure the Web Agent, the configuration wizard copies the LocalConfig.conf file to the following directory:

IIS web server

web_agent_home\bin\IIS

Sun Java System web server

Sun_Java_System_home/https-*hostname*/config

Apache web server

Apache_home/conf

The Web Agent retrieves its configuration settings from this copy of the LocalConfig.conf file.

Parameters Found Only in Local Configuration Files

For central Agent configurations, most of the parameters in the local configuration file are also in an Agent Configuration Object. The following parameters are used in the local configuration file only and are *not* found in Agent Configuration Objects:

AgentConfigObject

Defines the name of an Agent Configuration Object (stored on a policy server) in a local agent configuration file. This parameter is *not* used in Agent Configuration Objects.

Default: no default

EnableWebAgent

Activates a Web Agent and allows it to communicate with the Policy server. Set this parameter to yes only after you have finished changing all of the configuration parameters.

Default: No

HostConfigFile

Specifies the path to the SMHost.conf file (in an IIS 6.0 or Apache agent) that is created after a trusted host computer has been successfully registered with a Policy server. All Web Agents on a computer share the SMHost.conf file.

Default: No default

How to Edit an Agent Configuration File

The agent configuration file controls the settings of a locally-configured Web Agent. To change those settings, use the following process:

1. Create a backup copy of WebAgent.conf (for a traditional agent) or LocalConfig.conf file (for a Framework agent).
2. Open the original copy of the agent configuration file with a text editor.
3. Enable or disable parameters by doing any of the following:
   - Removing the pound sign (#) from the beginning of the line to enable a parameter.
   - Adding the pound sign (#) to the beginning of the line to disable a parameter.
4. Change the values of parameters using the following guidelines:
   - Do not add spaces between the parameter names, the equals sign (=), and the parameter values.
   - Surround the parameter values with quotation marks.
   - The WebAgent.conf and LocalConfig.conf files are not case-sensitive. You do not have to match the case shown in the sample file installed with the Agent.
   - Many values are shown in the file as descriptive variables, such as <Agent Name>,<IPAddress>. Replace the angle brackets and text with the values you want.
   - In cases where the value is Empty, a blank is valid as the default. A default value applies only if there is no pound sign (#) preceding the parameter.
5. Set EnableWebAgent to yes only when you are done making changes, then save and close the file.

   This makes all local configuration changes effective. If you make more changes after an Agent has been enabled, you must restart your web server for those changes to become take effect.

Implement Local Configuration

You can control whether local configuration is allowed with the following parameter:

AllowLocalConfig

Instructs the Agent Configuration Object on the Policy Server to read the local configuration file to obtain configuration parameters for the Web Agent. This parameter is used only in Agent Configuration Objects.

You can also add multiple values for this parameter in the Agent Configuration Object to control which parameters can be changed in a local configuration file. When multiple values are set for this parameter, they are processed in the following order:

- If the value of yes exists together with other configuration parameters in the same Agent Configuration Object, yes takes precedence. The other configuration parameters are the only ones that can be changed in the local configuration file.
- If the value no exists together with other configuration parameters in the same Agent Configuration Object, no takes precedence. This lets you quickly disable local configuration entirely without having to remove any of the other configuration parameters from the Agent Configuration Object.
- If the multiple values of yes and no exist in the same Agent Configuration Object, no takes precedence. This lets you quickly disable local configuration entirely without having to remove any of the other configuration parameters from the Agent Configuration Object.

Default: No

Example: yes, EnableAuditing, EnableMonitoring (allows local control of the only the two previous parameters).

To implement local configuration

1. Log into the Administrative UI.

   The Welcome screen appears.

2. Click the Infrastructure, Agent Configuration.

   A list of agent configuration tasks appears.

3. Click Modify Agent Configuration.

   The search window appears.

4. (Optional) Fill out the search form to narrow your search criteria.

5. Click Search.

   A list of Agent Configuration objects appears.

6. Click the radio button on the left of the Agent Configuration Object associated with the Agent on which you want to implement local configuration, and then click Select.

   The Modify Agent Configuration dialog appears.

7. Click the arrow to the left of the AllowLocalConfig parameter.

   The Edit Parameter dialog appears.

8. Change the text in the Value field to yes, and then click OK.

   The Edit Parameter dialog closes.

9. Click Submit.

   A confirmation message appears. Local configuration is enabled.

10. Open the appropriate local configuration file on your web server and change the parameter settings you want.

11. For traditional agents only, set the value of the EnableWebAgent parameter to yes.

12. Save and close the local configuration file.

13. For Framework agents only, do the following steps:

    a. Open the WebAgent.conf file.

    b. Set the value of the EnableWebAgent parameter to yes.

    c. Save and close the WebAgent.conf file.

14. Restart the web server.

    Local configuration is enabled and any updated parameters are changed.

More information:

Parameters Requiring a Server Restart when Changed (see page 25)
Enable a Web Agent (see page 65)

Restrict Changes to Local Configuration Parameters

With central agent configuration, you can restrict the configuration parameters that can be set by a local web server administrator. You may want to do this if the SiteMinder administrator in your organization is not the same person who is responsible for the web server where the SiteMinder Agent is installed.

To restrict changes to local configuration parameters

1. Log into the Administrative UI.

   The Welcome screen appears.

2. Click the Infrastructure, Agent Configuration.

   A list of agent configuration tasks appears.

3. Click Modify Agent Configuration.

   The search window appears.

4. (Optional) Fill out the search form to narrow your search criteria.

5. Click Search.

   A list of Agent Configuration objects appears.

6. Click the radio button on the left of the Agent Configuration Object you want. Click Select.

   The Modify Agent Configuration dialog appears.

7. Click the arrow to the left of the AllowLocalConfig parameter.

   The Edit Parameter dialog appears.

8. Change the text in the Value field to yes, and then click the Multi-value radio button.

9. Click Add.

   An empty field appears.

10. Type the name of the parameter to which you want to allow access in the field. Only those parameters in the list can be changed locally.

11. (Optional) Repeat Steps 9 and 10 to add more parameters.

12. Click OK.

    The Edit Parameter dialog closes, and the Modify Agent Configuration dialog appears.

13. Click Submit.

14. The Modify Agent Configuration dialog closes, and a confirmation message appears. Your changes will be applied the next time the Web Agent polls the Policy server.

Central and Local Configuration Together

If you have a large number of Web Agents that you want to configure centrally, but the settings of a few of those Web Agents need to be different than the others, you can use a combination of central and local configuration together.

For example, if you need to configure multiple cookie domain single sign-on across a SiteMinder network without configuring the Agents individually, you can use a central configuration for all of the agents, and local configuration settings for the smaller group that needs the different settings.

In the previous example, suppose the CookieDomain parameter in the Agent Configuration Object is set to example.com. However, for one Web Agent in your network, you want to set the CookieDomain parameter to .example.net, while still using all the other parameter values set in the Agent Configuration Object.

To implement the example configuration

1. With the Administrative UI, create an Agent Configuration Object with all the parameters that you want for your environment. Set the CookieDomain parameter to .example.com
2. Set the AllowLocalConfig parameter of the Agent Configuration Object to yes.
3. At one Web Agent, change *only* the local configuration file (on the web server) to use example.net as the value of the CookieDomain parameter. Do *not* modify any other parameters.

The value for the CookieDomain parameter in the lone Agent's local configuration file overrides the value in the Agent Configuration Object, while the Agent Configuration Object determines the settings for all the other parameters.

Chapter 2: Use Cases

This section contains the following topics:

How to Meet the Use Case Deployment Prerequisites

All of the Web Agent use cases have the following prerequisites, that you must complete before deploying a Web Agent:

1. Use the Administrative UI to do the following:
   - Define the resources you want to protect.
   - Implement an authentication scheme.
2. Select a configuration method for each Web Agent. Choose *one* of the following:
   - Central Configuration
   - Local Configuration
   - Central and Local Configuration together.
3. If you plan to use both traditional and framework agents together in the same environment, you may want to review the server platforms and Web Agent versions that apply to each Type of Web Agent.

More information:

Web Agent Configurations (see page 27)
Types of Web Agents (Traditional and Framework) (see page 23)

Use Case 1: Single Agent Protects Single Resource

This use case describes how to configure one Web Agent to protect a single resource, such as a website or application. The environment for this use case consists of the following components and settings:

- One Web Agent
- One web server

How to Implement Use Case 1

To implement Web Agent use case one, use the following process:

1. Ensure you have met the prerequisites.
2. Set the Agent Name and Default Agent Name Identities (see page 48).
3. Implement any of the following logging types:
   - Error logging (see page 164)
   - Trace logging (see page 167)
   - The OneView montior (see page 52)
4. Enable the Web Agent (see page 65).

Use Case 2: Multiple Agents Protecting Multiple Applications within One Domain

This use case describes how to configure multiple Web Agents to protect a several resources within one domain. The environment for this use case consists of the following components and settings:

- Several Web Agents
- Several web servers
- Several Resources
- Single sign-on
- A Cookie Provider

How to Implement Use Case 2

To implement Web Agent use case two, use the following process:

1. Ensure you have met the prerequisites.
2. (Optional) Configure any virtual servers (see page 68).
3. Set the Agent Name and Default Agent Name Identities for each Web Agent (see page 48).
4. Enable single sign-on (see page 83).
5. Configure a cookie provider (see page 86).
6. Implement any of the following functions:
   - Error logging (see page 164)
   - Trace logging (see page 167)
   - The OneView montior (see page 52)
7. Enable the Web Agents (see page 65).

Use Case 3: Framework and Traditional Agents Protecting Multiple Applications within One Domain

This use case describes how to configure a combination of Framework and Traditional Web Agents to protect a several resources within one domain. The environment for this use case consists of the following components and settings and domains:

- Several Framework and Traditional Web Agents
- Several web servers
- Several Resources
- Single sign-on
- A Cookie provider

How to Implement Use Case 3

To implement Web Agent use case three, use the following process:

1. Ensure you have met the prerequisites.
2. (Optional) Configure any virtual servers (see page 68).
3. Set the Agent Name and Default Agent Name Identities for each Web Agent (see page 48).
4. Use the following procedures to ensure proper compatibility between your traditional and framework agents:
   - Ignore the Cookie Provider for Unprotected Resources (see page 100)
   - Ignore the Cookie Provider for POST Requests (see page 101)
   - Use FCCs and NTCs in a Mixed Environment (see page 236)
5. Enable single sign-on (see page 83).

   Note: Set the SecureURLs parameter for all of the agents in your SSO environment to the same value.
6. Configure a cookie provider (see page 86).
7. Implement any of the following functions:
   - Error logging (see page 164)
   - Trace logging (see page 167)
   - The OneView montior (see page 52)
8. Enable the Web Agents (see page 65).

Use Case 4: Framework and Tradtional Agents Protecting Multiple Applications Across Multiple Domains

This use case describes how to configure a combination of Framework and Traditional Web Agents to protect a several resources across multiple domains. The environment for this use case consists of the following components and settings:

- Several Framework and Traditional Web Agents
- Several web servers
- Several Resources
- Single sign-on
- Cookie Provider configured
- (Optional) SSO Zones used

How to Implement Use Case 4

To implement Web Agent use case four, use the following process:

1. Ensure you have met the prerequisites.
2. (Optional) Configure any virtual servers (see page 68).
3. Set the Agent Name and Default Agent Name Identities for each Web Agent (see page 48).
4. Use the following procedures to ensure proper compatibility between your traditional and framework agents:
   - Ignore the Cookie Provider for Unprotected Resources (see page 100)
   - Ignore the Cookie Provider for POST Requests (see page 101)
   - Use FCCs and NTCs in a Mixed Environment (see page 236)
   - Enable Post Preservation between Framework and Traditional Agents (see page 245)
5. Enable single sign-on (see page 83).

   Note: Set the SecureURLs parameter for all of the agents in your SSO environment to the same value.
6. Configure a cookie provider (see page 86).
7. (Optional) Implement SSO Zones (see page 282).
8. Implement any of the following functions:
   - Error logging (see page 164)
   - Trace logging (see page 167)
   - The OneView montior (see page 52)
9. Enable the Web Agents (see page 65).

Chapter 3: Server-Specific Web Agent Configuration

This section contains the following topics:

Default Settings of Web Agent Configuration Parameters

The default settings for the Web Agent configuration parameters are always used unless a different value is specified.

If a parameter does not exist in the Agent Configuration Object or local configuration file, the default value is used.

More information:

Agent Configuration Parameters (see page 307)

Set the Agent Name and Default Agent Name Identities

The Agent name is the Agent's identity. The Policy Server uses this identity to tie policies to a Web Agent. You can define the name of a Web Agent with the following parameters:

AgentName

Defines the identity of the Web Agent. It establishes a mapping between the name and the IP address of each web server instance hosting an Agent.

If a value is not set for this parameter, or if the Web Agent does not find a match among the values listed, the Web Agent uses the value set in the DefaultAgentName parameter instead.

Note: This parameter can have more than one value. Use the multi-value option when setting this parameter in an Agent Configuration Object. For local configuration files, add the parameter name followed by each value to separate lines in the file.

Default: No default

Limits: Must contain 7-bit ASCII characters in the range of 32-127, and include one or more printable characters. Cannot contain the ampersand (&) and asterisk (*) characters. Not case-sensitive. For example, the names MyAgent and myagent are treated the same.

Example: myagent1,192.168.0.0

Example: myagent, www.sitea.com

DefaultAgentName

Defines a name that the Web Agent uses when it receives a request on an IP address or interface for which there is no agent name specified in the AgentName parameter.

If you are using virtual servers, you can set up your SiteMinder environment quickly by using a DefaultAgentName instead of defining a separate Web Agent for each virtual server.

Important! If you do not specify a value for the DefaultAgentName parameter, you must list every agent identity in the AgentName parameter. Otherwise, the Policy Server will not be able to tie policies to the Web Agent.

Default: No default

Limits: Must contain 7-bit ASCII characters in the range of 32-127, and include one or more printable characters. Cannot contain the ampersand (&) and asterisk (*) characters. Not case-sensitive. For example, the names MyAgent and myagent are treated the same.

If you are configuring virtual server support, you must specify a value for the AgentName or DefaultAgentName parameter.

To set the agent name and default agent name identities

1. Specify the agent name identities by doing either of the following:
   - For central agent configurations, open the Agent Configuration Object on the Administrative UI, and then add the values that you want to the AgentName parameter.
   - For local agent configurations, open the local configuration file on your web server. Add the values you want on a separate lines in the file.
2. Specify a default agent name identity by doing either of the following:
   - For central agent configurations, open the Agent Configuration Object on the Administrative UI, and then add the value that you want to the DefaultAgentName parameter.
   - For local agent configurations, open the local configuration file on your web server. Add the value you want to the DefaultAgentName parameter.

   The agent name and default agent name identities are set.

More Information

How to Set Up Virtual Server Support (see page 68)

Ensure that Agent Names Match

SiteMinder rules and policies are tied to Agent names. If a request is made to a host with an Agent name that is unknown to the Policy Server, the Policy Server cannot implement policies. Therefore, the value for the Web Agent's DefaultAgentName or AgentName parameter must match the name of an Agent entry defined at the Policy Server.

You define an Agent at the Policy Server using the Administrative UI. The value you enter in the Name field of the Agent Properties dialog box is the value that must match the name defined for the DefaultAgentName or AgentName setting, whether the Web Agent is configured locally (Agent configuration file) or centrally from the Policy Server (Agent Configuration Object).

Encrypt the Agent Name

The Web Agent, by default, adds its name to the URL that redirects a user to a forms, SSL, or NTLM credential collector. You can control whether the Agent encrypts its name in the URL and whether the credential collector decrypts the name when it receives the URL with the EncryptAgentName parameter.

The default setting for the EncryptAgentName parameter is yes. You should set this parameter to no in either of the following situations:

- If a third-party application is working with the credential collector and it must be able to read the Agent name for processing.
- If you configure a Web Agent as a Forms Credential Collector (FCC) for forms authentication, and direct users to a single resource to be authenticated. The procedure to configure a single resource target requires an un-encrypted Agent name.

To encrypt the Web Agent name, set the EncryptAgentName parameter to yes.

More Information

Configure the FCC to Use a Single Resource Target (see page 240)

How to Manage Web Agent and Policy Server Communication

You can manage the communication between a Web Agent and the Policy Server using any of the following procedures:

- Use CA Wily Introscope to monitor Web Agents.
- Use the SiteMinder OneView monitor for auditing purposes.
- Specify how often the Agent retrieves policy changes.
- Accommodate network latency issues.
- Manage Web Agents with Multiple Web Server instances.

Use CA Wily Introscope to Monitor Web Agents

If you are already using CA Wily Introscope in your organization, you can monitor the health of your SiteMinder Web Agents with the following parameter:

EnableIntroscopeApiSupport

Collects information about the SiteMinder Web Agent and sends it to CA Wily Introscope using a plug-in. This parameter uses the following settings:

- When set to yes, the Wily plug-in calls an API to collect the data.
- When set to no, the Wily plug-in creates an HTTP header with the data.
- When set to both, the Wily plug-in calls the API *and* creates an HTTP header with the data.
- When set to none, data not collected.

Default: No.

Limits: Yes, Both, No, None.

Example: (HTTP header) sm-wa-perf-counters = *server_name*.example.com:6180,86117203,86118343,1,0,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1125,0,15,1,1,750,750,

To use CA Wily Introscope to monitor the health of your Web Agents, set the value of the EnableIntroscopeApiSupport parameter to *one* of the following:

- Yes
- Both
- No

Monitor Web Agents with the OneView Monitor

The SiteMinder OneView monitor reports cache statistics and other information to the Policy Server, which administrators can use to analyze and fine-tune the Web Agent. You control the SiteMinder OneView monitor with the following parameter:

EnableMonitoring

Specifies whether the SiteMinder Web Agent sends monitoring information to the Policy Server.

Default: No

To have the Web Agent use the SiteMinder OneView Monitor, set the EnableMonitoring parameter to yes.

Note: For more information, see the Policy Server documentation.

Change How Often an Agent Checks for Policy or Key Updates

The Web Agent polls the Policy Server at regular intervals to check for the following items:

- Updated management information
- Updated policies
- Dynamically updated agent keys

You can change this interval according to your needs with the following parameter:

PSPollInterval

Specifies how often (in seconds) the Web Agent contacts the Policy Server to retrieve information about policy changes or dynamically updated keys. Higher numbers (longer intervals) decrease network traffic. Lower numbers (shorter intervals) increase network traffic.

Default: 30

Limit: 1

Example: If the level of traffic in your network is moderate, or your website does not change often, you can increase the interval to reduce network traffic.

To change how often a Web Agent checks the Policy Server for updates, change the number of seconds in the PSPollInterval parameter.

More information:

Web Agents and Dynamic Key Rollovers (see page 22)

Accommodate Network Latency

When network latency issues exist, the Web Agent cannot connect with the Policy Server. To avoid this problem, you can use the following parameter in the Agent Configuration Object or local configuration file:

AgentWaitTime

Specifies the number of seconds that the Web Agent waits for the Low-level Agent Worker process (LLAWP) to become available. When the interval expires the Web Agent tries to connect to the Policy Server.

Setting this parameter can help to resolve agent start-up errors related to LLAWP connections. We recommend starting with the default value and then increasing the interval by 5 seconds at a time until the agent starts successfully.
If you do not want to set this parameter in the Agent Configuration Object or LocalConfig.conf file, you can also set it in the WebAgent.conf file instead.

Default: 5

Example: If you have primary and secondary policy servers, try starting with value from 30 through 40.

Limit: None

Note: This parameter was originally developed for IIS Web Agents, but you can use it with other Framework Agents (on supported Windows platforms) if you experience network latency issues. Use the higher setting only if network latency issues exist. A high setting may cause unexpected web server behavior.

To accommodate network latency, add the AgentWaitTime parameter to your Agent Configuration Object or local configuration file, and then specify the number of seconds you want.

Manage Web Agents with Multiple Web Server Instances

If you configure a Web Agent on multiple web server instances, each server instance should have its own Web Agent cache, log file, and health monitoring resources. To ensure resources are unique, you configure the following parameter:

ServerPath

Specifies a unique path to each web server instance when a Web Agent is configured to use multiple instances of a web server. The ServerPath creates a unique identifier for the Web Agent's caching, logging, and health-monitoring resources.

Default: Empty

Example: If there are four web server instances, each loading a Web Agent, then each server's WebAgent.conf file should have the ServerPath parameter set to a unique value. You can set the ServerPath parameter to the directory where the web server's log file is stored, such as *server_instance_root*/logs.

Note: Do not add this setting to the LocalConfig.conf file for IIS 6.0 or the LocalConfig.conf file for Apache 2.0 servers (if there is only one instance of the Apache server)

To configure a Web Agent on multiple server instances, add a unique path to the ServerPath parameter.

Set the ServerPath Parameter for Windows Systems

The ServerPath parameter is required for Apache web servers on Windows if there are multiple server instances and you do not want these instances to share the same Agent resources (cache, logging, health monitoring). To help ensure that web server instances have their own resources, specify a ServerPath value.

The value you set for the ServerPath parameter must be an alphanumeric string unique among server instances running on the system. For example, if you have two server instances, the value of the ServerPath parameter for one instance could be MyAgent1 and the value for the second instance could be MyAgent2.

Note: Do not use a backslash (\) in the string you specify for the ServerPath parameter; all other characters are permitted.

The ServerPath parameter is *not* required for the following Windows platforms:

- IIS 6.0 servers (there is always only one server instance)

- Apache 2.0 servers if there is only one web server instance. The parameter is supported on these systems, but it serves no purpose.
- The Sun Java Systems or Domino web servers.

  These servers do not use the ServerPath parameter because it does not run in multiprocess mode on Windows.

Set the ServerPath Parameter for UNIX Systems

The ServerPath parameter is located in the WebAgent.conf file. Do *not* add this setting to the LocalConfig.conf file for framework Agents.

For web servers on UNIX platforms, we recommend that each server instance has its own Agent resources.

Set the ServerPath parameter for the following servers on UNIX:

- Any Apache 2.0 (including any Apache 2.0-based servers, such as IHS) and Apache 1.x web server instance
- Any Sun Java System web server instance

Note: The ServerPath is not required for Domino web servers on UNIX systems.

The value you set for the ServerPath parameter must be an alphanumeric string unique among the server instances running on the system. For example, if you have two server instances, the value of the ServerPath parameter for one instance could be MyAgent1 and the value for the second instance could be MyAgent2.

Additional Configurations Requiring the ServerPath Parameter

Situations that may require the ServerPath to be set are as follows:

- The Web Agent tracks shared memory using a semaphore. A semaphore is a value in operating system (or kernel) storage that each process can check to see if a resource is available or not, and then change. The semaphore is not generated uniquely, so multiple Agents would try to point to the same area of memory, causing the Agent to function improperly. Naming a server path gives the root of an instance, which allows the Web Agent to find the files used for creating unique keys for semaphores.
- For multiple server instances (all platforms except Windows), the Agent fails to execute one of the following:
  - To encrypt agent name (00-0012 error)
  - To encrypt SMSESSION or SMIDENTITY cookies
  - To get agent key updates at start up
- With Apache (all platforms except Windows), the Agent does not release its six shared memory segments (semaphores) when Apache is restarted.
- If each Web Agent is configured for a different web server type on the same system, such as an Apache 2.0 server and an IIS 6.0 server, you are required to specify a unique ServerPath value for each server's configuration. Different web server types cannot share Agent resources.

Chapter 4: Server-Specific Web Agent Configurations

This section contains the following topics:

Special IIS Web Agent Settings

You may need to make additional configuration changes to your IIS Web Agent if you have any of the following situations or conditions in your environment:

- Your resource requests return with 404 not found errors
- You use compact P3P Policies
- You use the IIS 6.0 Security Context
- You need to limit the size of your POST data

Manage 404 Not Found Errors (IIS 6.0 Agent)

If a Web Agent on an IIS 6.0 web server returns a 404 Not found error in response to a resource request, or if the Web Agent is not being called properly by the IIS 6.0 web server, open the IIS console and ensure that Wildcard Application mapping has been enabled and is configured to refer to the Web Agent DLL. Also, ensure that the Verify file exists check box is cleared before the mapping is enabled.

Configure your Web Agent to Accommodate P3P Compact Policies

You can determine whether the custom responses from your Web Agent comply with P3P response headers with the following parameter:

P3PCompactPolicy

Determines whether custom responses comply with the Platform for Privacy Preferences Project (P3P) response headers. P3P compact policies use tokens representing the specific elements from the P3P terminology. If you set the P3PCompactPolicy parameter to the appropriate policy syntax, it ensures that custom responses are set with the correct P3P response header when a P3P compact policy is specified for the Web Agent.

Default: No default

Example: NON DSP COR CURa TAI (these represent: none, disputes, correct, current/always, and tailoring, respectively)

To accommodate P3P compact policies, add an appropriate policy syntax to the P3PCompactPolicy parameter.

Limit Size of Post Data (IIS 5.0 Agents only)

Starting with SiteMinder 5.x QMR 2, IIS 5.0 Web Agents observe a 64 KB data size limit when preserving or filtering POST data. This restriction is mandated by known issues with New Atlanta Communications ServletExec. You can remove this restriction with the following parameter:

DisablePostDataLimit

Specifies whether a Web Agent observes the 64 KB data-size limit when preserving or filtering POST data. This does not affect the standard POST operation, but it does affect the following:

- POST preservation
- eTelligent Rules Post variables
- TransactionMinder authentication schemes

Default: No (limit enforced)

Important! Change this parameter to yes at your own risk.

Note: This parameter is not supported for framework Web Agents.

To remove the 64 KB data size limit restriction, set the DisablePostDataLimit parameter to yes.

Enable the IIS 6.0 Security Context to Work with the Agent

The SiteMinder Web Agent on an IIS 6.0 web server functions as an ISAPI extension. When an HTTP request is made, the IIS 6.0 web server challenges the user before the Web Agent responds to the request. The IIS server uses its native authentication scheme, Basic authentication, for the authentication challenge, if that scheme is selected in the server's Management Console.

To enable the IIS 6.0 security context to work with the agent

1. Open the IIS Management Console, and expand the local computer and Web Site folders.
2. Right-click the folder of your website, and then select Properties.

   The properties dialog (for your website) appears.
3. Click the Directory Security tab. In the Authentication and Access section, click Edit.

   The Authentication Methods dialog appears.
4. Ensure that Enable anonymous access check box is selected, and then click OK.

   The Authentication Methods dialog closes.
5. Click OK.

   The properties dialog closes. The IIS 6.0 Security Context is enabled to work with the SiteMinder Web Agent.

Remove the Server HTTP Header if Using the URLScan Utility

If you want to use the URLScan utility from Microsoft to remove the Server HTTP Header from the responses your IIS Web server sends, you also need to set the following parameter for your IIS Web Agent:

SuppressServerHeader

Prevents an IIS Web Agent from returning the Server HTTP Header in its responses. When the value of this parameter is set to no, the Web Agent sends the Server header with its responses and the IIS Web server passes it along to the client. When the value of this parameter is set to yes, the web agent does not send the Server header in its responses.

Default: No

The URLScan utiltiy removes the header from the IIS server's responses, while hte SuppressServerHeader parameter removes the header from the Web Agent's responses. Both the utility and the parameter must be set to prevent the Server header from being sent to the client in all responses.

To keep the Web Agent from sending the Server header in responses, set the value of the SuppressServerHeader parameter to yes.

Special Apache Web Agent Settings

You may need to make additional configuration changes to your Apache Web Agent if you have any of the following situations or conditions in your environment:

- Applications that redirect traffic to specific web servers *without* modifying the HTTP headers.
- Legacy applications (before HTTP 1.1) that do *not* support transfer-encoding.
- The Web Agent is logging excessive informational messages to the Apache error log

Use the HTTP HOST Request for the Port Number

If you have applications that perform load balancing by redirecting traffic to specific web servers *without* modifying the actual HTTP headers, you should configure the Web Agent to redirect users back to the proper external port (instead of the port used by the load balancer) with the following parameter:

GetPortFromHeaders

Directs the Web Agent to obtain the port number from the HTTP HOST request header instead of obtaining it from the web server service structures.

Default: No

Note: This parameter is required for Apache Web Agents.

To use the port number in the HTTP HOST request header, set the GetPortFromHeaders parameter to yes.

Use Legacy Applications with an Apache Web Agent

If you have legacy applications (that do not support HTTP 1.1), and you want to run them on an Apache Web Server, you can set the following parameter:

LegacyTransferEncodingBehavior

Specifies the type of message encoding used by the Web Agent. When the value of this parameter is set to no, transfer-encoding is supported.

When the value of this parameter is set to yes, content encoding is used. The transfer-encoding header is ignored and only the content-length header is supported.

Default: No

To use legacy applications with an Apache Web Server, set the value of the LegacyTransferEncodingBehavior parameter to yes.

Important! If you set the value of this parameter to yes, these features will not work: Federation; preservation of POST data longer than 4 KB; and large certificates may not be recognized.

Restrict IPC Semaphore-Related Message Output to the Apache Error Log

By default the Apache Web Agent logs all levels (informational and error) of IPC semaphore-related messages to the Apache error log, regardless of the configured Apache logging level.

To restrict the verbosity of Web Agent IPC semaphore-related output to the Apache error log, add the following parameter in the trace.conf file located in *web _agent_home*/config:

nete.stderr.loglevel

Specifies the level of IPC semaphore-related messages the Web Agent logs to the Apache error log. Accepts the following values:

off

The Web Agent logs no IPC semaphore-related messages to the Apache error log.

error

The Web Agent logs only IPC semaphore-related error messages to the Apache error log.

info

(Default) The Web Agent logs IPC semaphore-related error and informational messages to the Apache error log.

Example: Define the nete.stderr.loglevel parameter in trace.conf

In the following snippet from trace.conf, the nete.stderr.loglevel parameter is configured to restrict the Web Agent to log only IPC semaphore-related *error* messages to the Apache error log:

```
# CA Web Agent IPC logging levels
# nete.stderr.loglevel=error
```

Restrict Directory Browsing on a Sun Java System Server

To help ensure that users who try to browse the directories of a Sun Java System web server are challenged by SiteMinder, you can set the following parameter:

DisableDirectoryList

Specifies whether the Web Agent allows a user to view or browse the contents of a directory without challenging them first. This occurs when *all* of the following conditions are true:

- The realm is set to protect a root resource (/)
- The default web page in the directory (such as index.html) is renamed or deleted.

Default: No

To restrict directory browsing on a Sun Java System server

1. Add the DisableDirectoryList parameter to your Agent Configuration object or your local configuration file.
2. Set the value of the DisableDirectoryList parameter to yes.

   Directory browsing is restricted. SiteMinder challenges users who try to browse directories.

More information:

Web Agent Processes Client Requests Twice (see page 305)

Chapter 5: Starting and Stopping Web Agents

This section contains the following topics:

Enable a Web Agent

Configure your Web Agent parameters and then enable the Web Agent to protect the resources on the web server.

Note: *No* resources are protected until you also define policies in the SiteMinder Policy Server.

To enable a Web Agent

1. Open the WebAgent.conf file.
2. Change the value of the EnableWebAgent parameter to yes.
3. Save and close the WebAgent.conf file.
4. Restart the web server if you changed the settings for any of the following parameters in your Agent Configuration Object or local configuration file:
   - AgentConfigObject
   - CacheAnonymous
   - HostConfFile (IIS 6.0 and Apache agents only)
   - MaxResourceCacheSize
   - MaxSessionCacheSize
   - PostPreservationFile
   - ResourceCacheTimeout

   The Web Agent is enabled.

Disable a Web Agent

If you want to stop the Web Agent from protecting the resources on your web server and stop communicating with the Policy Server, you must disable the Web Agent.

To disable a Web Agent

1. Open the WebAgent.conf file.
2. Change the value of the EnableWebAgent parameter to no.
3. Save and close the WebAgent.conf file.
4. Restart the web server if you changed the settings for any of the following parameters in your Agent Configuration Object or local configuration file:
   - AgentConfigObject
   - CacheAnonymous
   - HostConfFile (IIS 6.0 and Apache agents only)
   - MaxResourceCacheSize
   - MaxSessionCacheSize
   - PostPreservationFile
   - ResourceCacheTimeout

   The Web Agent is disabled.

Chapter 6: Configure Virtual Servers

This section contains the following topics:

How to Set Up Virtual Server Support

A virtual server is a logical entity that you configure on a physical server. This logical entity acts as an independent server. Virtual servers let you host multiple websites on one physical server. For example, using virtual servers, you could set up a server to host both www.mysite.com and www.yoursite.com.

You can assign any of the following to a virtual server:

- A unique IP address
- An IP address that is shared with the physical server
- An IP address that is shared with another virtual server

Although you configure only one Web Agent per web server, you can configure Agent identities to protect your virtual servers. If one user accesses the server through www.mysite.com and another user accesses the server through www.yoursite.com, each server is protected by an agent identity. The advantage of creating an agent identity for each virtual server is that you can define unique realms and rules for each site.

The settings that you define for the Web Agent apply to all virtual servers that you define for that web server instance; however, each virtual server processes requests independently and the Policy Server treats each virtual server request separately. For more information about virtual servers and how to configure them, see the documentation for your web server.

To configure support for virtual servers, do *one* of the following tasks:

- Define and add an Agent identity for each virtual server, specify a value for the AgentName parameter, and assign it the IP address or host header name of a virtual server.
- Define an Agent identity only for virtual servers that need to be uniquely identified.
- Set a Default Agent Name.

Note: If you have more than one instance of the Sun Java System web server, such as a server for HTTP communication and a server for HTTPS communication, two WebAgent.conf files exist. Each file can have multiple agent identities. (The name Sun Java System refers to the web server that was formerly called Sun ONE and iPlanet.)

More information:

Set the Agent Name and Default Agent Name Identities (see page 48)

Add a SiteMinder Wildcard Mapping to Protect IIS 6.0 Virtual Web Sites

SiteMinder automatically protects only the Default Web Site folder of the IIS web server. If you are running virtual web sites on your IIS 6.0 web server, add wildcard mappings to *each* virtual web site that you want to protect. After adding the wildcard mapping to the virtual sites, enable the Web Agent.

To add a SiteMinder wildcard mapping to protect IIS 6.0 virtual web sites

1. Configure virtual servers for the IIS 6.0 web server.

   Note: For more information, see your IIS documentation.
2. Open the IIS Management Console.
3. Right-click on the Virtual Web Site, and then select Properties.

   The Properties dialog appears.
4. Click the Home Directory tab.
5. Click Configuration.

   The Application Configuration dialog appears.
6. In the Wildcard application maps section, click Insert.
7. Click Browse and navigate to the following file:

   web_agent_home\SiteMinder Web Agent\Bin\ISAPI6WebAgent.dll

   web_agent_home

   Indicates the directory where the Web Agent is installed.

   Default (Windows installations): C:\Program Files\CA\webagent

   Default (UNIX installations): /opt/ca/webagent
8. Click OK twice.

   The Application Configuration and Properties dialogs close.
9. Restart the virtual website.
10. Repeat Steps 3 through 9 for each virtual web site you want to protect.

    The IIS Web Agent is ready to be enabled.

Assign Web Agent Identities for Virtual Servers

Additional Web Agents for each virtual server are not actually *defined*, but are *assigned* a Web Agent identity. To protect virtual servers that have unique access requirements or to protect distinct realms, assign each server a unique Agent identity and use the default agent name for all other virtual servers. The advantage of this option is that you can configure your SiteMinder installation quickly, yet still guard virtual servers hosting realms that require separate protection.

The AgentName parameter and its associated IP address provide mapping for web server interfaces to agent names as defined in the policy store. Web Agents need to make Agent API calls in the proper agent name context in order for the correct set of rules and policies to apply. If no Agent name or IP address is assigned for mapping to the policy store, then the Web Agent uses the value of the DefaultAgentName parameter only for a virtual server.

To protect virtual servers using unique Agent identities, add a Web Agent for each virtual server in the AgentName parameter. Adding separate Web Agents for each virtual server lets you define unique realms and rules for each virtual server.

To assign a Web Agent identity

1. Enter the name of the agent and the IP address, separated by a comma.
2. Specify the port number associated with the IP address (for example: 112.12.12.1:8080) if your virtual servers share the same IP address, but use different ports. If you are using default ports, port numbers are not required.
3. To add more than one Agent, put each entry on a separate line, as in the following example:

   ```
   agentname="agent1,123.123.12.12:8080"
   agentname="agent2,123.123.12.12:8081"
   agentname="agent3,123.123.12.13"
   ```

4. If you add an Agent Identity, you must define it in the Administrative UI with the same configuration. Make sure that the Agent Identity is defined in Administrative UI exactly as it is defined for the Agent configuration.

If it finds no entries in the AgentName parameter, SiteMinder uses the value of the DefaultAgentName only for a virtual server.

Note: If you change the DefaultAgentName, make sure that it is defined in the Administrative UI exactly as it is defined for the Agent.

Specify Virtual Servers for the Web Agent to Ignore

If a web server at your site supports several virtual servers, there may be resources on these virtual servers that you do not want to protect with the Web Agent. To simplify how the Web Agent distinguishes which portions of a web server's content it protects, use the following parameter:

IgnoreHost

Specifies the fully qualified domain names of any virtual servers that you want the web Agent to ignore. Resources on such virtual servers will be auto-authorized, and the Web Agent always grants access to them regardless of which client makes the request. The authorization decision is based on the configuration of the Web Agent instead of being based on a policy.

The list of ignored hosts is checked first before any other auto-authorization checks, such as the IgnoreExt and IgnoreURL settings. Therefore, the double-dot rule will not trigger an authorization call to the Policy Server for resources on an ignored host but would not be ignored by extension.

The host portion of the URL entries for the IgnoreHost parameter must exactly match what the Web Agent reads for the host header of the requested resource.

Note: This value is case-sensitive.

If the URL uses a specific port, then the port must specified.

For centrally-managed agents, use a multi-value parameter in the Agent Configuration Object to represent several servers. For agents configured with a local configuration file, list each host on a separate line in the file.

Example: (URL shown with port specified)

IgnoreHost="myserver.example.org:8080"

Example: (local configuration file)

IgnoreHost="my.host.com"

IgnoreHost="your.host.com"

Default: No default

To specify virtual servers for the Web Agent to Ignore, do either of the following tasks:

- For central configuration, add the servers you want to ignore to your agent configuration object. For more than one server, use the multi-value setting for the parameter.
- For local configuration, add a separate line for each server in the local configuration file.

Resources using the specified URLs are ignored by the Web Agent and access to those resources is granted automatically.

More Information

Handle Complex URIs (see page 217)

Resolve Agent Identity by IP Address

On virtual web servers, when IP addresses and host names are used to resolve the Agent name, the Web Agent can potentially use an incorrect value for AgentName to evaluate the request. This situation would allow unauthenticated users to access protected resources.

You can force the Web Agent to resolve the Agent name based on the physical IP address of the virtual server, with the following parameter:

UseServerRequestIp

Instructs the Web Agent to resolve the AgentName according to the physical IP address of a virtual web server. Use this parameter to increase security if a web server uses IP addresses for virtual server mappings. If this parameter is set to no, the Web Agent resolves the AgentName according to the host name in the HTTP Host header of the client's request.

For Domino servers, this parameter is supported only for Domino 6.x. If this parameter is enabled for an Agent on other Domino versions, the Web Agent uses the default Agent name.

For IIS Web Agents configured for SSL communication and virtual hosts, you must set this parameter to yes. IIS does not allow virtual host mappings using host names with SSL enabled.

Default: No

To resolve a Web Agent's identity using the IP Address, set the UseServerRequestIp parameter to yes.

Chapter 7: Use Platform for Privacy Preferences (P3P) Compact Policies with SiteMinder Web Agents

This section contains the following topics:

How to Support a P3P Compact Policy with your SiteMinder Web Agent

CA SiteMinder supports P3P Compact policies on most types of Web Agents, *except* for the following types:

- Apache 1.3x
- Domino

Note: For more information about P3P, see the P3P page of the World Wide Web Consortium web site.

To configure your Web Agent to support a P3P Compact policy, use the following process:

1. Configure a P3P Compact policy on your web server.

   Note: For more information, see the documentation provided by your web server vendor.

2. Configure your Web Agent to accommodate your P3P Compact policy.

Configure your Web Agent to Accommodate P3P Compact Policies

You can determine whether the custom responses from your Web Agent comply with P3P response headers with the following parameter:

P3PCompactPolicy

Determines whether custom responses comply with the Platform for Privacy Preferences Project (P3P) response headers. P3P compact policies use tokens representing the specific elements from the P3P terminology. If you set the P3PCompactPolicy parameter to the appropriate policy syntax, it ensures that custom responses are set with the correct P3P response header when a P3P compact policy is specified for the Web Agent.

Default: No default

Example: NON DSP COR CURa TAI (these represent: none, disputes, correct, current/always, and tailoring, respectively)

To accommodate P3P compact policies, add an appropriate policy syntax to the P3PCompactPolicy parameter.

Chapter 8: Single Sign-On (SSO)

This section contains the following topics:

How Single Sign-on Works in a Single Domain

SiteMinder provides single sign-on functionality across single and multiple cookie domains. This simplifies using applications across different Web servers and platforms, and improves the user experience because the users do not have to re-authenticate as they move across a single sign-on environment.

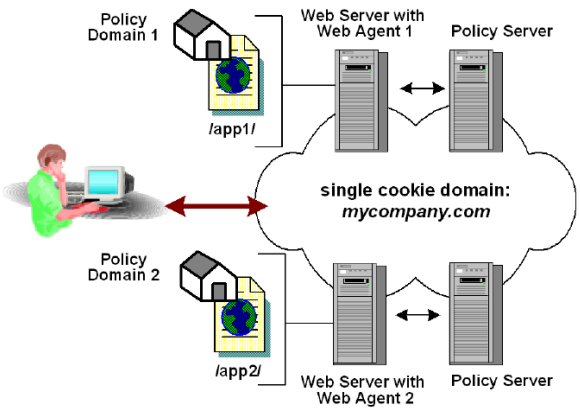
A single domain is an environment where all resources exist in the same cookie domain. Multiple Web Agents in the same cookie domain can be configured for single sign-on if you specify the same cookie domain in each Web Agent's configuration.

If single sign-on is enabled, it uses the following process:

1. The user authenticates once.
2. The Web Agent caches the successful authentication, and then issues a single sign-on cookie to the user's browser.
3. The single sign-on cookie provides the session information, so that users can access the following types of resources without reauthenticating:
   - Protected resources in other realms with an *equal* or *lower* protection level
   - Another web server within this cookie domain

   Users who try to access resources with a *higher* protection level must re-authenticate before they are granted access.

The following illustration shows single sign-on in a single cookie domain:

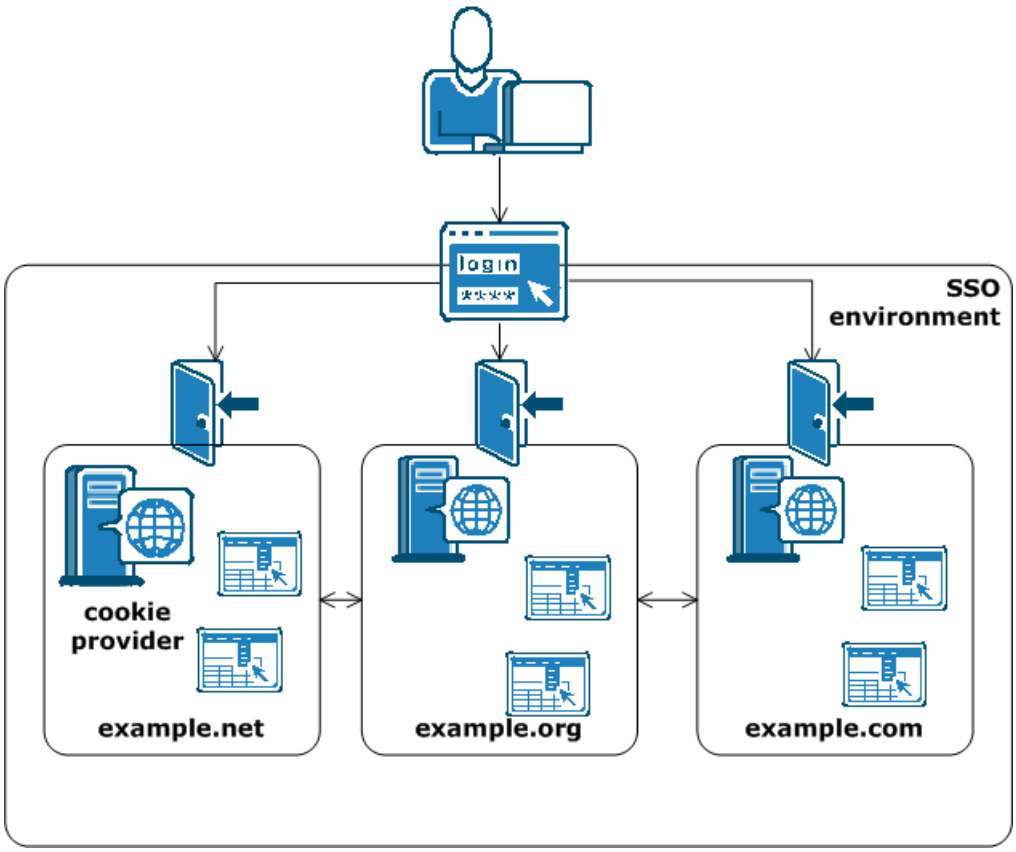


Note: If you are using replicated user directories with non replicated policy stores, the user directory must be named identically for all policy stores. Also, the session ticket key, which encrypts session tickets, must be the same for all key stores in the SSO environment. The session ticket determines the duration of a valid user session.

Single Sign-On Across Multiple Domains

Without single sign-on, users are often required to log on and enter their credentials multiple times as they access different applications and resources on separate servers in different cookie domains. The ability to pass single sign-on information across multiple cookie domains enables a user to authenticate at a site in one cookie domain, and then go to a site in another cookie domain without being rechallenged for credentials. For the user, this seamless navigation makes related sites easier to use.

The following illustration shows single sign-on across multiple cookie domains.



Single Sign-On Across Multiple Cookie Domains

SiteMinder implements single sign-on across multiple cookie domains using a SiteMinder Web Agent configured as a cookie provider.

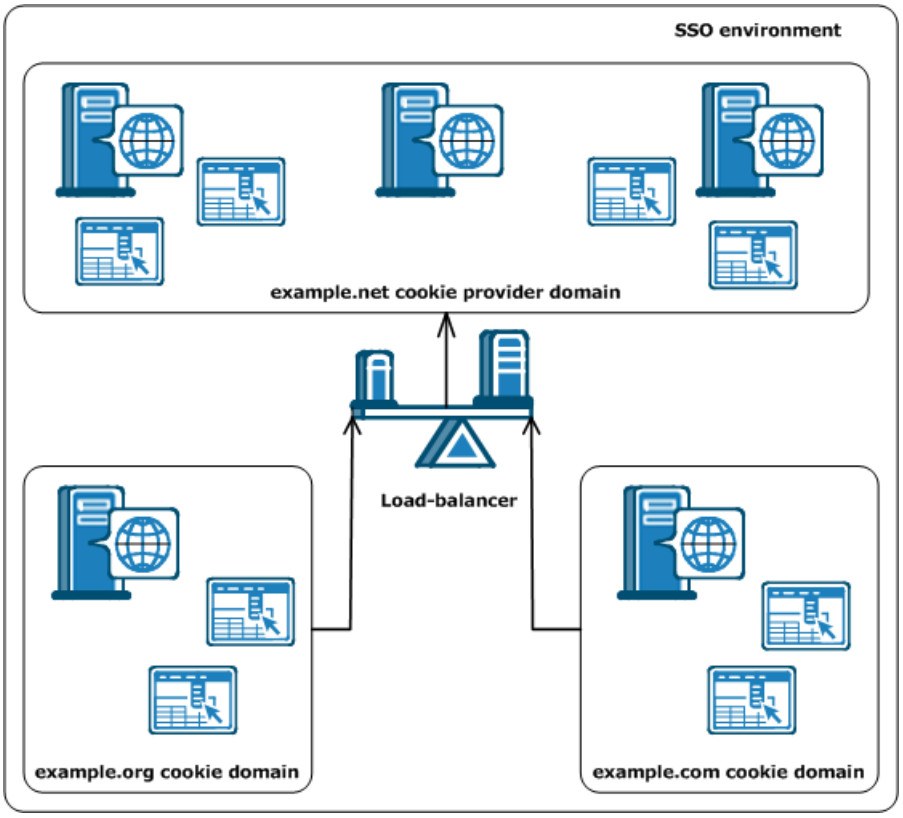
The cookie domain where the cookie provider Web Agent resides is named the cookie provider domain. All the other Web Agents from the other cookie domains within the single sign-on environment, point to one cookie provider.

SiteMinder cookie providers work using the following process:

1. A user requests a protected resource in a domain within the single sign-on environment, and is challenged for credentials.
2. When the user is authenticated, the following cookies are set in the user's browser:
   - the local cookie for the domain where the user has authenticated
   - the cookie set by the cookie provider.
3. The user can navigate between the domains in the single sign-on environment without being rechallenged until either of the following events occur:
   - The user's session times out
   - They end the session (usually by closing their browser)

Will the Web Agents in your single sign-on environment need to be load-balanced?

Because all Web Agents in an SSO environment must refer to a single cookie provider domain, add a load-balancer between the web servers in your cookie provider domain and the other cookie domains in your SSO environment as shown in the following illustration:



The Web Agent in the example.org cookie domain points and the Web Agent in the example.com cookie domain both point to the same cookie provider domain of example.net. A load-balancer distributes the traffic evenly between all the web servers in the example.net cookie provider domain.

Note: SSO across multiple cookie domains does not require that the same user directory be used across the SSO environment. However, if you are using replicated user directories with non-replicated policy stores, the user directory must be named identically for all policy stores. Also, the session ticket key, which encrypts session tickets, must be the same for all key stores in the SSO environment. The session ticket determines the duration of a valid user session.

Single Sign-On and Authentication Scheme Protection Levels

With single sign-on, authenticated users of one realm can access a resource in another realm without re-authenticating as long as the second realm is protected by an authentication scheme with an equal or lower protection level. If a user tries to access a resource protected by an authentication scheme with a higher protection level, SiteMinder prompts the user to re-enter their credentials.

SiteMinder lets administrators assign protection levels to authentication schemes with the Administrative UI. Protection levels range from 1 through 20, with 1 being the least secure and 20 being the most secure. These protection levels enable administrators to implement authentication schemes with an additional measure of security and flexibility for a single sign-on environment.

For example, a set of resources that is available to all users has a Basic authentication scheme with a protection level of 1. Another set of resources that should only be available to corporate executives, uses an X.509 certificate scheme with a protection level of 15. If a user authenticates with the Basic theme, then tries to access the resources protected by a certificate scheme, they will be required to re-authenticate.

Note: For more information, see the Policy Server documentation.

Allow Automatic Access to Resources that use the OPTIONS Method

The SiteMinder Web Agent still challenges authenticated users who attempt to access resources that use the OPTIONS method. Some examples of resources that use the OPTIONS method include (but are not necessarily limited to) the following:

- Microsoft® Word documents
- Microsoft® Excel® spreadsheet documents

This challenge occurs because the application associated with the resource sends a request using the OPTIONS method to the web server. Because this request does *not* include a SiteMinder cookie, the Web Agent issues a challenge.

To prevent users from being challenged for these resources

1. Set the value of the following parameter to yes:

   autoauthorizeoptions

   Automatically authorizes any requests for resources which use the HTTP OPTIONS method.

   If you set the value of this parameter to yes, also set the value of the PersistentCookies parameter to no.

   Limits: yes, no

2. Set the value of the PersistentCookies parameter to no.

Track User Identity Across Anonymous Realms

When an anonymous user accesses resources, that user is assigned an SMIDENTITY (anonymous) cookie. When the user moves to another domain, the user is challenged, logs in successfully, and is assigned an SMSESSION (logged in) cookie.

As this user accesses protected and "anonymous" resources, that is, resources in a realm that do not require a user to present credentials, the user may enter a domain that contains both cookies for a user. For resources protected by Web Agents starting at 5.x QMR 3 , the Web Agent uses the SMSESSION cookie to identify the user, not the SMIDENTITY cookie.

If the user goes from a thoroughly upgraded domain to a domain where older Agents use the SMIDENTITY cookie to identify the user, the cookie used depends on the version of the Web Agent handling the request.

Regarding separate cookie domains, when a master cookie domain contains protected resources and a second domain contains anonymous resources, a user who does the following tasks continues to be treated as an anonymous user in the anonymous domain:

1. Accesses the anonymous domain first
2. Moves to the master domain and logs in
3. Moves back to the anonymous domain

Single Sign-on and Agent Key Management

Web Agents use keys to encrypt and decrypt cookies that pass information between Web Agents. When an Agent receives a SiteMinder cookie, the key allows the Agent to decrypt the contents of the cookie. Keys must be set to the same value for all Web Agents communicating with a Policy Server.

To ensure the keys remain secure, the Policy Server can generate these keys, encrypt them, and distribute them to all the Web Agents within a SiteMinder environment. Automated key changes make agent key management easy to implement for large SiteMinder installations that share the same key store, which holds all the key information. Automating key changes also ensures the integrity of the keys.

How to Configure Single Sign-On

To set up your single sign-on environment, use the following process:

1. Decide which cookie domains comprise the single sign-on environment.
2. Decide which cookie domain within the single sign-on environment will be the cookie provider domain.
3. Access the Agent Configuration Object (using the Administrative UI) or open the Web Agent configuration file (on your web server), and modify the parameters in the following steps:
   a. Set the RequireCookies parameter to yes.

      If you set the timeout parameters without requiring cookies, the Web Agent functions normally; however, it cannot enforce the timeouts. If the Web Agent requires cookies but the user's browser does not accept them, the user will be denied access to all protected resources.

      Note: For more information, see the Policy Server documentation.

   b. If you want the cookies to last until the configured session timeout, set the PersistentCookies parameter to yes.

      If you set this parameter to no, the cookies last for only one browser session.

   c. For the CookieDomain parameter, verify that this is the local cookie domain of the system on which the Web Agent is installed, such as .mycompany.com. Modify the domain, if necessary. This value is case-sensitive.

   d. Set the CookieProvider parameter to the cookie provider domain using the appropriate syntax as follows:

      http://*server.domain:port*/siteminderagent/SmMakeCookie.ccc

      where *server.domain:port* is the fully qualified domain name of the Web Server where the Web Agent acting as the cookie provider resides, such as myserver.mysite.com. The cookie provider name must have a .ccc extension.

   e. Make sure the cookie provider is configured with the proper associated MIME type (.ccc).

   f. Enable the proper type IP checking (to compare IP addresses) if you enabled persistent or transient cookies.
4. (Optional) Modify any other single sign-on settings.
5. If you edited parameters by modifying the Web Agent configuration file, restart the web server, so that the changes take effect.

More Information

Configure MIME Types for Each Credential Collector (see page 233)
Compare IP Addresses to Prevent Security Breaches (see page 227)

Require Cookies for Basic Authentication

You can control whether or not SiteMinder requires cookies with the following parameter:

RequireCookies

Specifies whether SiteMinder requires cookies. SiteMinder uses cookies to do the following:

- Secure a single sign-on environment
- Track session timeouts
- Track idle time-outs.

Important! If you configure the Web Agent to require cookies, a user's Web browser must accept HTTP cookies. If the browser does not, the user receives an error message from the Agent denying the user access to all protected resources.

Default: Yes

RequireCookies is a special setting that is useful only if basic authentication was set during the Policy Server configuration. This setting instructs the agent to require either an SMSESSION or an SMCHALLENGE cookie in order to successfully process HTTP requests, including basic Authorization headers.

If the Web Agent does not require cookies, but the user's Web browser is accepting cookies, the Web Agent functions normally; however, the user may be challenged for credentials unexpectedly and the Web Agent may not strictly enforce time-outs.

To require cookies, set the RequireCookies parameter to yes.

Set Persistent Cookies

If you want to use single sign-on for multiple browser sessions, use persistent cookies. If you set persistent cookies, users can end their browser sessions before a SiteMinder session expires, start a new browser session, and still have single sign-on capability.

Beginning with Web Agent 5.x QMR1, cookies that are written to a client system's hard disk remain valid for the configured maximum session time-out *plus* 7 days. Typically, a persistent cookie is deleted from a Web browser's cookie file after the cookie expires; however, browsers may handle persistent cookies differently.

To set persistent cookies

1. Set the PersistentCookies parameter to yes

   The SMSESSION cookies will be persistent.

2. Set the TransientIDCookies parameter to no.

   The SMIDENTITY cookies will be persistent.

Specify the Cookie Domain

The CookieDomain parameter defines the cookie domain of the web sever where you installed the Web Agent, such as netegrity.com. You specify the cookie domain during the Web Agent installation.

You can modify the domain, if necessary. This value is case-sensitive. Note the following when setting this parameter:

- If you set CookieDomain to none, it forces the Web Agent to generate cookies only for the web server hosting the Web Agent. These are server-only cookies. For example, myserver.netegrity.com.
- If you leave CookieDomain blank or set it to double quote marks ("")—in the local configuration file, the Web Agent gets the cookie domain from the HTTP_HOST header then bases the value on the CookieDomainScope parameter.

  When the CookieDomainScope parameter is set to 0, the default, the Agent chooses the most specific cookie domain for the host without making a server-only cookie. This means that the cookie domain myserver.netegrity.com yields a domain of netegrity.com, and myserver.metals.ne.com yields a domain of .metals.ge.com. If the CookieDomainScope parameter is set to 2, the cookie domain would be .netegrity.com and .ne.com respectively.
- If you set the CookieDomain parameter to a specific domain, such as .netegrity.com, that is the domain the Web Agent uses.

Specify the Cookie Provider

To use a cookie provider, you must specify its location with the following parameter:

CookieProvider

Specifies the URL (using the fully qualified domain name) of the web server where the Web Agent that is acting as the cookie provider resides. The cookie provider name must have a .ccc extension.

- For IIS, Sun Java System and Domino web servers, the URL syntax is as follows:

  http://*server.domain:port*/siteminderagent/SmMakeCookie.ccc

- For Apache and Apache-based web servers, the URL syntax is as follows:

  http://*server.domain:port*/SmMakeCookie.ccc

This parameter affects the following parameters:

- CCCExt
- SessionUpdatePeriod

Default: No default

Example: (IIS, Sun Java System and Domino web servers)
http://server1.myorg.com:80/siteminderagent/SmMakeCookie.ccc

Example: (Apache and Apache-based web servers)
http://server1.myorg.com:80/SmMakeCookie.ccc

To specify the cookie provider

1. Set the CookieProvider parameter to the URL of the web server.
2. Make sure the CCCExt parameter is set to .ccc
3. (Optional) Modify the session update period.

   The cookie provider is specified.

More Information

Configure MIME Types for Each Credential Collector (see page 233)

Modify the Session Update Period

You can specify how often the Web Agent redirects a request to the Cookie Provider to set a new cookie with the following parameter:

SessionUpdatePeriod

Specifies how often (in seconds) a Web Agent redirects a request to the Cookie Provider to set a new cookie. Refreshing the master cookie decreases the possibility that it will expire due to an idle time-out of the SiteMinder session.

Default: 60

To modify the session update period

1. Make sure the CookieProvider parameter is defined.
2. Change the number of seconds in the SessionUpdatePeriod parameter to reflect the interval you want.

   The session update period is changed.

Enable Single Use Session Cookies

You can increase the security of your environment by having SiteMinder create session cookies that are only used once. Single-use session cookies prevent anyone with access to the following items from copying a session cookie and then re-using it to gain unauthorized access to resources:

- Web server logs
- SiteMinder Web Agent logs
- Potentially compromised proxy servers sitting between domains (in the case of cross domain single sign on)

You can control whether SiteMinder uses single-use or multiple-use session cookies by setting the following parameter:

StoreSessioninServer

Specifies whether single-use session cookies are used. When the value of the StoreSessioninServer parameter is yes, a single-use session cookie is created and stored on the session server. Cookie providers and Web Agents access the cookie from the session server.

Cookie providers and Web Agents replace the session cookie in a URL with a GUID that corresponds to the single-use session cookie stored on the session server.

When the value of the StoreSessioninServer parameter is no, the session cookie is passed directly in the URL

Default: No

To enable single use session cookies

1. Ensure your environment meets the following conditions:
   - Upgrade your Web Agents and cookie providers to use SiteMinder 6.0 SP5 QMR1 or higher.
   - Use a value for the DefaultAgentName parameter in the Web Agents and cookie provider.
   - Your Policy Server is configured with a valid session store.
2. In your Web Agents and cookie provider, set the value of the StoreSessioninServer parameter to yes.

Validate a Session Cookie Domain

You can reduce the risk that unauthorized users may hijack and attempt to reuse SiteMinder session cookies by having SiteMinder validate the domain of a session cookie with the following parameter:

TrackSessionDomain

Instructs the Web Agent to encrypt and store the intended domain of a session cookie within the session cookie itself. When the session cookie is presented for subsequent requests, The Web Agent compares the intended domain stored within the session cookie against the domain of the requested resource. If the domains do *not* match, the Web Agent rejects the request.

For example, when the value of this parameter is set to yes, session cookies intended for use with operations.example.com would be rejected by the Web Agent if they were presented at finance.example.com.

Default: No

To have SiteMinder validate the domain of a session cookie, set the value of the TrackSessionDomain parameter to yes.

Prevent Session Cookie Creation or Updates

Some Web applications, such as Microsoft Outlook Web Access, make HTTP requests behind the scenes even when a user is not actively using the application. For example, the Web Access application makes HTTP requests even when the user is not actively checking for new email on the server.

These requests may update the SMSESSION cookie so that the session never expires, even though the user has been idle. You can prevent the Web Agent from creating or updating session cookies during these background requests so that sessions expire normally.

To prevent creating or updating SMSESSION cookies

1. Configure one or both of the following parameters:

   OverlookSessionForMethods

   Specifies whether the Web Agent compares the request method of all HTTP requests against the methods listed in this parameter. If a match occurs, the Web Agent does not create or update an SMSESSION cookie. Also, cookie providers (if configured) are not updated for that request.

   Default: No default

   OverlookSessionForUrls

   Specifies whether the Web Agent compares the URLs from all HTTP requests against the URLs listed in this parameter. If a match occurs, the Web Agent does not create or update an SMSESSION cookie. Also, cookie providers (if configured) are not updated for that request.

   Default: No default

   Example: Use a relative URL, such as /MyDocuments/index.html. Do not use an absolute URL (http://fqdn.host/MyDocuments/index.html)

Note: If you configure both of the previous parameters, the methods are processed before the URLs.

Prevent Session Cookie Creation or Updates Based on Method and URI

Some Web applications, such as Microsoft Outlook Web Access, make HTTP requests behind the scenes even when a user is not actively using the application. For example, the Web Access application makes HTTP requests even when the user is not actively checking for new email on the server.

These requests update the SMSESSION cookie so that the session never expires, even though the user has been idle. You can prevent the Web Agent from creating or updating session cookies during these background requests so that sessions expire typically.

To prevent creation or updates based on method and URI

1. Set all the following parameters:

   OverlookSessionForMethods

   Specifies whether the Web Agent compares the request method of all HTTP requests against the methods listed in this parameter. If a match occurs, the Web Agent does not create or update an SMSESSION cookie. Also, cookie providers (if configured) are not updated for that request.

   Default: No default

   OverlookSessionForMethodUri

   Specifies whether the Web Agent compares the URI from all HTTP requests against the URI listed in this parameter. If a match occurs, the Web Agent does not create or update an SMSESSION cookie. Also, cookie providers (if configured) are not updated for that request.

   Default: No default.

   Note: Methods are processed before URIs.

Set Secure Cookies

You can specify that session cookies are only sent between a protected web server and the requesting browser over secure (HTTPS) connections using the following parameter:

UseSecureCookies

Sends cookies to web servers using secure (HTTPS) connections. Enable this parameter to increase security between browsers and web servers.

When this setting is enabled, users in single sign-on environments who move from an SSL web server to a non-SSL web server will have to reauthenticate. Secure cookies cannot be passed over traditional HTTP connections.

Default: No

To send cookies over SSL connections, set the UseSecureCookies parameter to yes.

More information:

Set Secure Cookies Across Multiple Domains (see page 93)

Set Secure Cookies Across Multiple Domains

Setting the UseSecureCookies parameter configures a Web Agent to only return a *local* cookie to a requesting browser session if the connection between them is secure (HTTPS); if the Web Agent is also configured as a cookie provider, UseSecureCookies does not apply to redirected requests for access to resources in other cookie domains.

To configure a Web Agent acting as a cookie provider to only return cookies to a Web Agent in another cookie domain if that Web Agent is also configured to use secure cookies, you must enable UseSecureCookies and also configure the following parameter:

UseSecureCPCookies

If UseSecureCPCookies is set to Yes, the cookie provider will only send a cookie to a Web Agent in another cookie domain that is also configured to use secure cookies (that is, UseSecureCookies is enabled).

When this setting and UseSecureCookies are both enabled, users in a multiple domain single sign-on environment who move from an SSL web server to a non-SSL web server in another cookie domain will have to reauthenticate. Secure cookies cannot be passed over traditional HTTP connections.

Default: No

To send cookies over SSL connections across multiple domains, set the UseSecureCookies and UseSecureCPCookies to yes on the cookie provider.

More information:

Set Secure Cookies (see page 92)

Control Identity Cookies

The TransientIDCookies parameter specifies whether or not the Agent identity cookie (SMIDENTITY) is transient or persistent.

Persistent cookies are written to a client system's hard disk. Prior to Web Agent 5.x QMR1, persistent cookies remained valid for 7 days. Beginning with Web Agent 5.x QMR1, persistent cookies remain valid for the configured maximum session timeout plus 7 days. (The maximum session timeout is set in the Administrative UI.) Typically, a persistent cookie is deleted from a Web browser's cookie file after the cookie expires; however, browsers may handle persistent cookies differently. By default, the Web Agent does not use persistent cookies. It uses transient cookies.

If you want to use single sign-on for multiple browser sessions, use persistent cookies. If you set persistent cookies, a user can end their browser session before a SiteMinder session expires then start a new browser session and still have single sign-on capability.

Whereas persistent cookies are written to a hard disk, transient cookies are not written to the hard disk and they are not subject to configured session time-outs. Transient cookies will remain in your cookie folder.

Set TransientIDCookies to no, if you want the identity cookie to be persistent. Leave the default value set to yes, if you want the identity cookie to be transient.

Be sure to set the corresponding IP Check.

More Information

Compare IP Addresses to Prevent Security Breaches (see page 227)

Modify the Session Grace Period

Web pages usually consist of many resources, all of which are potentially protected by the Web Agent. For each resource associated with a single request, a session cookie is generated. To eliminate the overhead of generating multiple session cookies for a single user request, set the following parameter:

SessionGracePeriod

Specifies the number of seconds during which a SiteMinder session (SMSESSION) cookie will not be regenerated. Cookies are not regenerated when *all* of the following conditions are met:

- There is no URL SMSESSION cookie.
- The difference between the current time and the last access time of the received SMSESSION cookie is less than or equal to the SessionGracePeriod.
- The amount of time between the current time and the time when the received cookie would have been idle exceeds two grace periods. For example, if your grace period is 25 minutes and the idle time-out is 60 minutes, SiteMinder regenerates a session cookie after 10 minutes, because then there are less than two grace periods (50 minutes) of time left before the session goes idle.

Default: 30

To modify the session grace period

1. Change the value of the SessionGracePeriod parameter.
2. If you increased the setting for the SessionGracePeriod parameter in Step 1, use the Administrative UI to ensure both of the following values in all of your realms do *not* exceed the value of the SessionGracePeriod parameter:
   - Session timeout value
   - Idle timeout value

   The session grace period is changed.

   Note: Session timeouts are part of configuring a realm, which you do using the Administrative UI. For further instructions on configuring session timeouts, see the Policy Server documentation.

Set a Time-out for Saved Credentials

When a user chooses to have credentials saved, the Policy Server instructs the Web Agent to create a persistent cookie containing the user's credentials. The cookie allows Web Agents to authenticate a user based on the credentials saved in the cookie, instead of challenging the user to authenticate. You can control how long the persistent cookie remains with the following parameter:

SaveCredsTimeout

Specifies the number of hours that a persistent cookie containing the user credentials will be saved. During this time interval, the Web Agent authenticates the user with the data stored in the cookie. After this time interval expires, the cookie is removed and the Web Agent challenges the user again.

Default: 720 (30 days)

To set a timeout for saved credentials, enter the number of hours you want in the SaveCredsTimeout parameter.

Note: For more information, see the Policy Server documentation.

How to Enforce Timeouts across Multiple Realms

User session timeouts are governed by the realm that the user first logs into. If a user enters a new realm through single sign-on, the time-out values for the new realm are still governed by the session that was established by the initial login at the first realm. If you have different time-out values for different realms, and you want to have each realm use its own time-out values, you can override the time-outs of the original realm.

A user who has already timed out cannot log in to another realm without being rechallenged. For example, if the Idle Timeout in Realm1 is 15 minutes and the Idle Timeout in Realm2 is 30 minutes, a user who accumulates 20 idle minutes in Realm1 will be challenged upon logging in to Realm2.

To override the time-outs of the original realm, configure your Web Agent and realms as described in the following process:

1. Set the value of the EnforceRealmTimeouts parameter to yes.
2. Use the Administrative UI to do the following tasks:
   a. For each realm where you want to supersede the original time-outs (any realm that SSO functionality allows the user to access), do the following:
      - To override the Maximum Timeout value, create a response using the WebAgent-OnAuthAccept-Session-Max-Timeout response attribute.
      - To override the Idle Timeout value, create a response using the WebAgent-OnAuthAccept-Session-Idle-Timeout response attribute.
   b. Bind each of the previous responses to an OnAuthAccept rule.

   Note: For information about creating responses, see the *Policy Server Configuration* Guide.

Redirect a User after a Session Time-out

Session time-outs are set when you configure a realm with the Administrative UI. When a user's SiteMinder session times out, the Web Agent does one of the following actions:

- Rechallenges the user for credentials
- Redirects the user to another URL

If a redirect URL is specified, the user is sent to that destination page. If the page is unprotected, the user is granted direct access to that page. If the page is protected, the user is challenged for credentials before being granted access to the page. If no redirection URL has been specified, the Web Agent rechallenges the user for credentials after a session time-out.

You can redirect users whose sessions time out to a URL with a customized web page, which explains why their session has been terminated and how they can reestablish it. For example, you can create a custom web page that displays a message such as, "You have been logged out automatically as a security precaution. Please login again to continue."

If the user is not redirected to another page after a session times out, SiteMinder challenges the user again. This may confuse users because they may not understand why they are being asked to reauthenticate.

To redirect users to different URLs after session time-outs

1. Add the following parameters to your Agent Configuration Object or your local configuration file:

   IdleTimeoutURL

   Specifies the URL where the Web Agent should redirect the user when the idle time-out for the session occurs.

   Example:
   http://example.mycompany.com/sessionidletimeoutpage.html

   Note: IdleTimeoutURL should only be used for non-persistent sessions; it has no effect if configured for persistent sessions.

   MaxTimeoutURL

   Specifies the URL where the Web Agent should redirect the user when the maximum time-out for the session occurs.

   Example: http://example.mycompany.com/maxtimeoutpage.html

   Default: No default

2. Enter one URL for each of the previous parameters. You can use the same URL for all of the parameters, or you may use different URLs for each.

If the idle timeout and maximum timeout values for a session (set in the Policy Server) occur at the same time and the IdleTimeoutURL and MaxTimeoutURL parameters are set, the user is redirected to the URL specified in the MaxTimeoutURL parameter when a time-out occurs.

Protect Session Cookies from Misuse with Validation Periods and Expired Cookie URLs

SiteMinder uses time-based session cookie parameters that can substantially reduce the possibility of a SiteMinder session cookie being compromised by administrators or other users who have access to the following items:

- Web server logs
- SiteMinder Web Agent logs
- Potentially compromised proxy servers sitting between domains in the case of cross-domain single sign-on

These time-based session cookie parameters add the concept of "born dates" to session cookies. Agents receiving a session cookie as a result of a redirect (URL session cookie) will look for the cookie born date name/value pair and compare this value with the value set for the CookieValidationPeriod configuration parameter. If the value of the born date and the CookieValidationPeriod parameter value exceed the current time, the cookie is rejected.

To protect session cookies from misuse, set the following parameters:

CookieValidationPeriod

Specifies the time period (in seconds) in which the receiving agent will accept the session cookie. After this time passes, the session cookie will not be accepted. If this field is not used or is set to zero, the session cookie expires when the Idle Timeout and Max Session Timeout values are met.

Default: Empty.

ExpiredCookieURL

(Optional) Specifies a URL that the agent redirects the user to after any session cookie has expired. If neither the born date nor the CookieValidationPeriod are configured, the agent ignores the settings and processes the cookie as usual (backwards compatibility).

Configure Support for SDK Third-Party Cookies

If you use non-SiteMinder Web Agents in your organization, you can configure them to support single sign-on with the following parameter:

AcceptTPCookie

Allows the Web Agent to accept session (SMSESSION) cookies created by third-party (non-SiteMinder) Web Agents. Third-party agents generate and read SMSESSION cookies using the SiteMinder SDK.

Default: No default

Note: For more information, see the SiteMinder Developer's documentation.

To allow the Web Agent to accept session cookies created by non-SiteMinder Web Agents, set the AcceptTPCookie parameter to yes.

Ignore the Cookie Provider for Unprotected Resources

Web Agents forward all requests to the cookie provider by default. If you have unprotected resources, you can reduce network traffic with the following parameter:

IgnoreCPForNotprotected

Prevents the cookie provider from being queried for unprotected resource requests. When this parameter is set to no, all requests are directed to the cookie provider by the Web Agent. For traditional (non-framework) Agents, a cookie provider must be configured for the value of this parameter to appear in the Web Agent log file.

Default: No

To prevent Web Agent from contacting the cookie provider when unprotected resources are requested, set the value of the IgnoreCPForNotprotected parameter to yes.

Ignore the Cookie Provider for POST Requests (Framework Agents Only)

The following parameter lets you use traditional agents as cookie providers in an environment that also has framework agents:

LegacyCookieProvider

Controls whether a framework agent sends a POST request to a cookie provider. When framework agents send a POST request to a traditional agent that is acting as a cookie provider, the redirected request becomes a GET instead and fails. When set to no, the framework agent sends the POST request to the cookie provider. When set to yes, the framework agent does not send the POST request to the cookie provider.

If you are using central agent configuration, you must add this parameter to your Agent Configuration Object. This parameter already exists in local configuration files.

Default: No (POST requests sent)

To use a traditional agent as a cookie provider with a framework agent, set the LegacyCookieProvider parameter to yes.

Force the Cookie Domain

Using fully qualified domain names helps ensure that cookies work properly. You can force the Agent to append its cookie domain to the host name in a URL request that meets either of the following conditions:

- The request does not specify a domain
- The request contains only an IP address.

To force the Agent to append its cookie domain

1. Set the ForceCookieDomain parameter to yes.
2. Set the ForceFQHost parameter to yes.

   The Web Agent will append its cookie domain to the host name when necessary.

Implement Cookie Domain Resolution

To implement automatic domain resolution, comment out the CookieDomain parameter or set it to none to cause the Web Agent to create cookies that are good only for the server from which they were issued.

You can further define the cookie domain by adding a value to the CookieDomainScope parameter. The scope determines the number of sections, separated by periods, that make up the domain name. (A domain always begins with a ".")

A CookieDomainScope value of 0 instructs the agent to use the most specific scope for a given host. A value of 1 (resulting, for example, in a cookie domain of .com) is not allowed by the HTTP specification. The value 2 instructs the agent to use the most general scope.

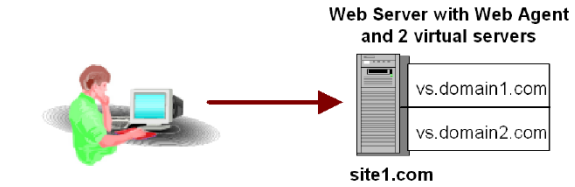
The following table shows some domain names and CookieDomainScope values.

Domain Name	Cookie Domain Scope value	Cookie Domain
server.myorg.com	2	.myorg.com
server.division.myorg.com	3	.division.myorg.com
	2	.myorg.com
server.subdivision.division.myorg.com	4	.subdivision.division.myorg.com
	3	.division.myorg.com
	2	.myorg.com

For example, the domain division.myorg.com has a scope of 3. By default, the Web Agent assumes a scope of 2; cookie domains cannot have a scope of 1.

Resolve Cookie Domains Automatically

You can configure the Web Agent to determine the cookie domain of an incoming request based on the request's incoming host header. Because the Agent can rely on the host header to determine the cookie domain, it does not require a static cookie domain value specified in its configuration. This flexibility enables you to configure virtual servers in different cookie domains for protection by the Web Agent, as shown in the following illustration:



Web Agent can respond to requests for:

- http://w1.site1.com/
- http://w2.vs.domain1.com/
- http://w3.vs.domain2.com/

Force Fully Qualified Domain Names

Cookies require the use of fully qualified domain names to work properly. The Web Agent can ensure that fully qualified domain names are used for HTTP requests regardless of the URL format a user enters in an incoming request. Enforcing the use of fully qualified domain names, along with the Agent's ability to resolve cookie domains based on incoming host headers, helps SiteMinder manage sites that have many domains across a network. Administrators can let users access a site with any valid form of a URL and still support cookies with the following parameter:

ForceFQHost

Forces a Web Agent to use a fully qualified domain name. This parameter uses configured domain name system (DNS) services to force the appending of the cookie domain to the host name in a URL request through DNS services and not an Agent. If the Web Agent receives a request that contains a partial URL, the Web Agent redirects the request back to the same destination resource specified in the original URI. The redirect request uses the fully qualified host name, which the Web Agent determines using the configured DNS services. Use this parameter with the ForceCookieDomain parameter for added functionality.

Default: No

Example: When the Web Agent receives a request from http://host1/page.html, it responds with http://host1.myorg.com/page.html. If the Web Agent receives a request such as http://123.113.12.1/page.html, it responds with http://host1.myorg.com/page.html.

Note: These examples work only if the proper DNS lookup tables are set up. If a partial domain is entered, the result depends on whether or not the DNS lookup can resolve it. If the request resolves as an invalid host, an error will result. Most likely, such a request would not even reach the web server.

To configure the Web Agent to use fully qualified domain names, set the ForceFQHost parameter to yes.

Modify the Cookie Domain

You should specify the cookie domain in the following situations:

- When configuring virtual servers in different cookie domains, you should not specify a cookie domain (with an empty value) because the Web Agent uses an incoming host header value in the user request to determine the cookie domain.
- When configuring a forms authentication scheme, specify the cookie domain of the authentication scheme's server name. For example, if the cookie domain is .myorg.com, the server name would be server1.myorg.com. We recommend using lower case values.

  Note: For more information, see the Policy Server documentation.

To modify the cookie domain, set the following parameter:

CookieDomain

Defines the cookie domain of the Web Agent that you specified during the Web Agent installation. This must be a fully qualified domain name with at least two periods. For example, the .myorg.com cookie domain matches the following servers:

- w1.myorg.com
- w2.myorg.com
- w3.sales.myorg.com

All web servers in this domain can exchange cookies with a user's browser. Servers in the same cookie domain use cookies to verify a user's credentials.

Default: Empty

Example: .mycompany.com

Note: This value is case-sensitive.

Configure SecureUrls with Single Sign-on

If your single sign-on network has a Web Agent that supports SecureUrls functionality and another Agent that does not, this could result in internal server error messages when a user requests a protected single sign-on resource.

The log for the Web Agent with SecureUrls support shows the reason for the server error, such as the following:

Error. Unable to process request, SecureUrls is disabled.

Note: All Web Agents in a single sign-on environment must have the SecureUrls parameter set to the same value. SiteMinder does not support interoperability between Web Agents with the SecureUrls parameter set to different values.

How Full Logoff Works

Full logoff support enables a Web developer to make sure that a user is completely logged off from a user session. This protects resources because it gives users a way to end a session without exiting the Web browser and prevents an unauthorized person from assuming control of an open session.

A full logoff uses the following process:

1. A user clicks a button to log off.
2. The Web Agent redirects the user to a customized logoff page that you created.
3. The Web Agent removes the session and authentication cookies from a user's browser.
4. The Web Agent also removes the session cookie from the local cookie domain and the cookie provider domain, which you specify for single sign-on environments.
5. The Web Agent calls the Policy Server and instructs the Policy Server to remove any session information.

   The user is completely logged off.

More information:

Configure Full Logoff Support for Domino Agents (see page 277)

Configure Full Logoff

Full logoff uses a custom logoff page that you create along with the following parameter:

LogOffUri

Enables full log off and specifies the location of a custom web page on your web server that appears to users after they are successfully logged off. You must configure this page so that it cannot be stored in a browser cache. Otherwise, a browser may display a logoff page from its cache without logging the user off. This may give an unauthorized user an opportunity to assume control of a session.

Note: When the CookiePath parameter is set, the value of the LogOffUri parameter must point to the same cookie path. For example, if the value of your CookiePath parameter is set to example.com, then your LogOffUri must point to example.com/logoff.html

Default: No default

Limits: Do *not* use a fully qualified URL.You must use a relative URI.

Example: /Web pages/logoff.html

To configure full logoff

1. Create a custom HTTP application to log the user off, for example, add an Exit or Sign Off button that redirects the user to a URL you specify.
2. To make sure that an HTML logoff page is loaded from the web server and not from the browser's cache, set up the logoff page so it cannot be cached in the browser. For example, for HTML pages, you can add the following meta tags to the page:

   ```
   <META HTTP-EQUIV="Pragma" CONTENT="no-cache">
   ```

   ```
   <META HTTP-EQUIV="Expires" CONTENT="-1">
   ```

   Important! Meta tags may not always work with an Internet Explorer browser. In this case, use a cache-control HTTP header.
3. Configure the LogOffURI parameter with the following steps:
   a. Delete the pound sign (#), if necessary.
   b. Enter the URI of the custom HTTP file that will log the user off. Do not enter a fully qualified URL.

      Full logoff is configured.

More information:

Specify the Cookie Path for Agent Cookies (see page 111)

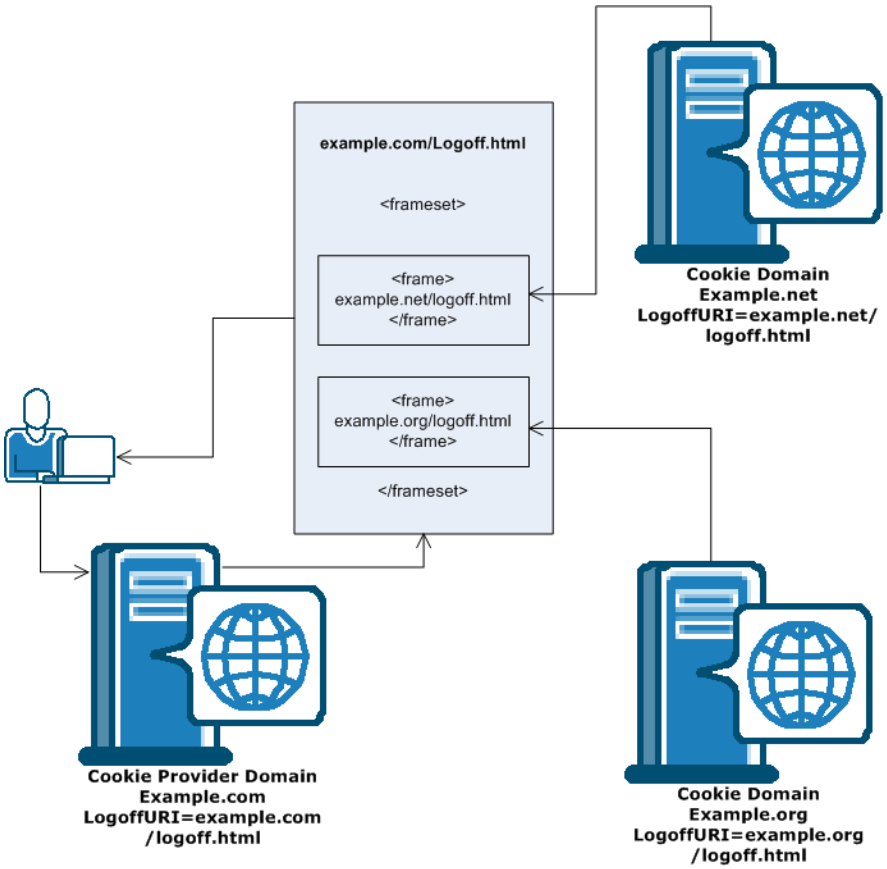
How to Configure Full Logoff for Single Sign-on

In a single sign-on environment, the session cookies are removed only from the local cookie domain and the cookie provider domain associated with the Web Agent. For single sign-on across multiple cookie domains, the full log-off feature of SiteMinder does *not automatically* log a user off across all the cookie domains that the user has visited.

To configure log-offs across multiple cookie domains, use the following process:

1. Create one centralized log-off page that contains separate frames (or iframes) for the other cookie domains in your SSO environment. These frames can be a small size, such as 1x1 pixels.
2. For each frame of the centralized log-off page in Step one, add a hyperlink to the Logoff URI of the associated cookie domain. For example, if you have two other cookie domains, example.org and example.net, you would do the following steps:
   - Add a hyperlink to the Logoff URI of example.org to one frame.
   - Add a hyperlink to the Logoff URI of example.net to the other frame.
3. Configure the LogoffURI of the cookie provider domain to point to the centralized log-off page. When the web server loads this log off page, the frames in the centralized log-off page call the logoff pages from the other cookie domains. The user is logged off from all the cookie domains at once.

The following illustration shows an example of using a centralized log-off page:



Note: You can also place the hyperlinks inside <iframe> tags instead of <frame> tags.

Integrate an IIS 6.0 Agent with SharePoint Portal Server 2003

The IIS 6.0 Web Agent can work with Microsoft's SharePoint Portal Server 2003 and provide single sign-on for resources stored on that server.

Note: In a SharePoint security context, the user store must be Active Directory (AD).

To integrate the Web Agent and the SharePoint sever,

1. Complete both of the following prerequisites:
   - Install a r12.0 SP2 (or later) Web Agent for IIS 6.0.
   - Apply the hotfix described in Microsoft Knowledge Base Article 824330, "IIS 6.0 Does Not Return AUTH_TYPE for Integrated Security."
2. Add the following to the default virtual server's web.config file, located, for example, in c:\inetpub\wwwroot2\web.config.
3. Place the following entry between the </system.web> attribute and the </configuration> attribute:

   ```
   <appSettings>
   <add key="SPS-EnforceIISAnonymousSetting" value="false"/>
   </appSettings>
   ```

4. Enable the session server so the Web Agent can provide the user security context for the SharePoint Portal Server resources.

   Note: For more information, see the Policy Server documentation.

   The Web Agent is integrated with the SharePoint Portal Server 2003.

Specify the Cookie Path for Agent Cookies

When a Web Agent creates a cookie, the web agent automatically uses the root (/) directory as the cookie path. The domain and path attributes of cookies are compared to the URL of a request. If the cookie is valid for the domain and the path, the client sends the cookie to the server. When the cookie path uses the root value, the client sends the cookie to the server with all requests in the domain.

You can set SiteMinder cookies to a given set of paths to eliminate the web traffic caused when cookies are sent for unprotected resources. For example, if a cookie path is set to /mypackage, the client only sends the cookie for requests in a particular package in the domain.

To specify the cookie path for agent cookies

1. Open your Agent Configuration Object or your local agent configuration file.
2. Set the Cookie Path for the Cookie Provider in the following parameter:

   MasterCookiePath

   Specifies the path for the primary-domain session cookies created by the cookie provider. For example, if this parameter is set to /siteminderagent, all session cookies that the cookie provider creates will have the /siteminderagent path. If this parameter is not set in the Cookie Provider Agent, the default value is used.

   Default: / (root)
3. Set the cookie path for the secondary agents in the following parameter:

   CookiePath

   Specifies the cookie path for the following secondary agent browser cookies:

   - xxSESSION
   - xxIDENTITY
   - xxDOMINODATA
   - xxCHALLENGE (including SSL_CHALLENGE_DONE)
   - xxDATA
   - xxSAVEDSESSION

   For example, setting this parameter to /BasicAuth, all of the secondary agents in the previous list are created using /BasicAuth as the path. If not specified, the default value is used.

   The CookiePath is *not* added to credential cookies (such as xxxxCRED) to maintain backwards compatibility with 4.x agents.

   The following cookies will *always* use the root (/) path:

- ONDENIEDREDIR
- TRYNO

If the CookiePathScope parameter is greater than zero, the CookiePath parameter settings are overriden.

Default: / (root)

4. (Optional) If you want the Web Agent to extract the cookie path from the URL instead of using the CookiePath value, set the following parameter to a number greater than zero:

CookiePathScope

Specifies the scope of the cookie path for the following secondary agent cookies:

- xxSESSION
- xxIDENTITY
- xxDOMINODATA
- xxCHALLENGE (including SSL_CHALLENGE_DONE)
- xxDATA
- xxSAVEDSESSION

Using a CookiePathScope greater than zero in this parameter overrides the setting of the CookiePath parameter.

Default: 0

More information:

Configure Full Logoff (see page 107)

How CookiePathScope Settings Work

The following table shows how the value of the CookiePathScope parameter works with the following settings:

- A URL such as http://*fqdn*/path1/path2/path3/path4/index.html
- A CookiePath parameter value of /BasicA

If the CookiePath value is:	And the CookiePathScope value is:	Then the following path is used:
/BasicA	0	/BasicA
/BasicA	1	/Path1

If the CookiePath value is:	And the CookiePathScope value is:	Then the following path is used:
/BasicA	2	/Path1/Path2
/BasicA	3	/Path1/Path2/Path3
/BasicA	4	/Path1/Path2/Path3/Path4
/BasicA	5	/Path1/Path2/Path3/Path4
/BasicA	99	/Path1/Path2/Path3/Path4
/ or "undefined"	0	/
/ or "undefined"	1	/Path1
/ or "undefined"	2	/Path1/Path2
/ or "undefined"	3	/Path1/Path2/Path3
/ or "undefined"	4	/Path1/Path2/Path3/Path4
/ or "undefined"	5	/Path1/Path2/Path3/Path4
/ or "undefined"	99	/Path1/Path2/Path3/Path4

These settings also affect simple SSO. For example, if the value of the CookiePathScope is set to 1 or higher,users will get challenged for credentials for both /BasicA/Index.html and /BasicB/Index.html since the SESSION cookie with a path /BasicA will not be valid for /BasicB/Index.html request.

Chapter 9: Protecting Web Applications

This section contains the following topics:

Mechanisms for Developing Web Applications

SiteMinder provides the following mechanisms for developing Web applications:

- Configurable response attributes

  Response attributes are sent by the Policy Server to the Web Agent to be included in HTTP headers or cookies. These are configurable by an administrator using the Administrative UI and they are available for the Web Agent.

- Default HTTP headers

  SiteMinder has a set of default HTTP headers that are passed from the Agent to Web applications.

- Custom error handling

  SiteMinder allows you to make error information relevant to your application by customizing HTML text in standard HTTP errors or using custom error pages.

More Information

How Response Attributes Work with Web Agents (see page 116)
SiteMinder Default HTTP Headers (see page 122)

How Response Attributes Work with Web Agents

SiteMinder response attributes instruct applications how to collect user data and apply that information to display personalized content for each user.

SiteMinder provides configurable response attributes as a means of delivering data to applications and customizing the user experience.

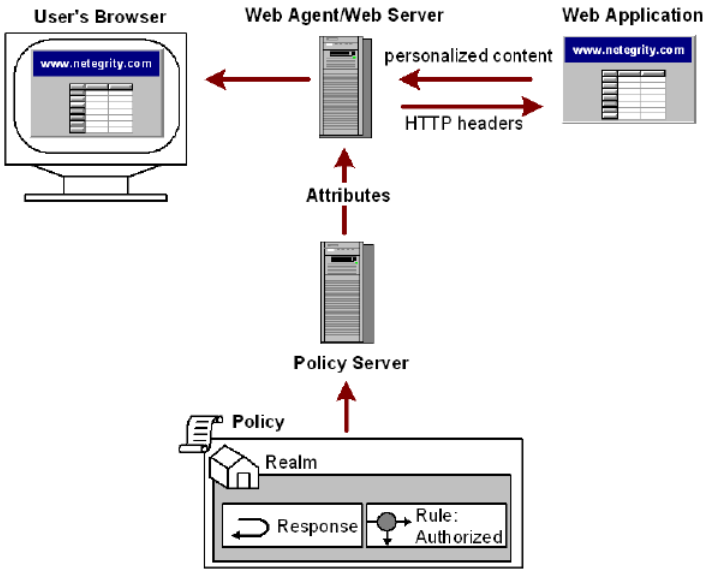
You configure responses using the Administrative UI, and then associate them with specific rules in a policy. When a request triggers a rule with a configured response, the Policy Server sends the response data to the Agent, which interprets the information and makes it available to Web applications.

When you configure a response, you associate the response with an Agent action. You can associate HTTP header and cookie response attributes with the actions GET and POST. These attributes can also be tied to Authentication or Authorization events. The Policy Server can send a response if the user is accepted or rejected for either of these events.

Note: When configuring response attributes note that the maximum buffer size for the web server for agent responses is 32 KB. There is no length limit of a response other than the total buffer size.

Response attributes other than the header and cookie attributes can *only* be used when an authentication or authorization event occurs (whether or not the user is accepted or rejected for either of these events). For example, you can select an **Authorization event** action for a rule and then configure a **WebAgent-OnReject-Redirect** response attribute. If a user is rejected during the authorization process by SiteMinder, the Agent redirects the user to another page that could display a message indicating why that user was rejected.

The following illustration shows how response attributes are sent from the Policy Server to the web server:



To simplify the task of maintaining responses, define a separate response for each type of event. For example, define one response for an OnAccept event and another response for an OnReject event. Creating a separate response makes it easier to find attributes when you need to modify response values.

HTTP Header and Cookie-Variables

The HTTP-Header-Variable and HTTP-Cookie-Variable attributes enable a Web Agent to pass a static or dynamic list of name/value pairs to a Web application. The name/value pairs are specific to the user requesting a resource, which enables the application to customize what the user sees.

For example, an administrator configures the WebAgent-HTTP-Header-Variable response attribute to store the full name of the user. When the user is authorized to access the protected resource, the Web Agent passes the user's full name to the Web application. The user's name is then displayed by the application, which helps to establish a relationship with the customer.

Be aware that in a Web application environment, the HTTP-Header-Variable response attribute appears as an HTTP_*attribute_name* variable, where *attribute_name* is the name of the HTTP variable, for example USERFULLNAME. You do not have to have an underscore in the name as the underscores cause problems with some application servers.

Note: The server may convert any dash (-) in the attribute name to an underscore (_), and all alphabetic characters to uppercase.

Cache Response Attributes

You can instruct a SiteMinder Agent to cache response attributes or expire attributes that contain dynamic data, forcing the Agent to contact the Policy Server and update the information. If you configure a static response attribute, the Policy Server only allows you to cache the value. By definition, static values do not change, so there is no need to recalculate them. If you configure user, DN, or active attributes, you can either cache the value or recalculate the value at specific intervals to ensure that the data is current.

How to Pass the Authenticated User Name to Applications

The REMOTE_USER CGI environment variable holds the name of the user authenticated by the web server. When the Web Agent is installed on a web server, SiteMinder replaces the native authentication of the web server, so REMOTE_USER is usually blank.

If you have applications that require the REMOTE_USER variable, you need to ensure that the REMOTE_USER variable is populated.

However, for web servers where REMOTE_USER cannot be populated or is not required, the Web Agent's default HTTP header, HTTP_SM_USER provides an alternate method of passing a user name to an application.

More Information

Configure the Web Agent to set the REMOTE_USER Variable (see page 120)

How the IIS Web Agent Populates the REMOTE_USER Variable

For an IIS web server to populate its REMOTE_USER header, Basic authentication has to be enabled for the web server. Basic authentication is set in the IIS Management Console in the Directory Security settings.

When Basic authentication is enabled and a user requests a SiteMinder-protected resource, the Web Agent attempts to set the IIS web server's HTTP_Authorization header by providing a user name but not a password. The presence of the HTTP_Authorization header means that the IIS server's Basic authentication takes precedence over any other authentication challenge. Therefore, the IIS web server thinks that the user is responding to its own challenge. Unless an ISAPI filter, such as the SiteMinder Web Agent sets the user context of the request, the IIS web server attempts to authenticate the user name passed by the incomplete HTTP_Authorization header.

Because the Web Agent operates as an ISAPI filter, it can set the user context of the request and provide a value for the REMOTE_USER header. The Agent populates the REMOTE_USER header based on the SetRemoteUser parameter being set to Yes plus the configuration of any one or more of the following Web Agent parameters:

- DefaultUsername and DefaultPassword—together these parameters control the (privileged) proxy user account used by the Web Agent for most activities.
- ForceIISProxyUser—overrides the normal behavior and forces the Web Agent to instruct the IIS web server to run as the proxy user.
- UseAnonAccess—instructs the Web Agent to provide no user context for the request at all, leaving any existing user context unchanged.
- Run in Authenticated User's Security Context—overrides the normal behavior because the Web Agent instructs the IIS web server to use the credentials stored in the persistent session.

Be cautious when using the SetRemoteUser parameter and the UseAnonAccess parameter together.

The following table shows how these parameters work together.

If...	Then...
SetRemoteUser=yes and UseAnonAccess=yes	The REMOTE_USER variable cannot be set because the Web Agent does not pass along a user security context. The lack of a user security context forces the IIS web server to use the credentials from the HTTP_Authorization header that the Agent modified; however it is incomplete because it contains only the user name.
SetRemoteUser=yes and UseAnonAccess=no	The Web Agent can pass along a user context of some type, depending on how other parameters are set, such as DefaultUserName, DefaultPassword, or ForceIISProxyUser. If the Web Agent does pass on a security context, the IIS web server ignores the incomplete HTTP_Authorization header in favor of the credentials provided by the Web Agent.

Configure the Web Agent to set the REMOTE_USER Variable

Configure the Web Agent to set the REMOTE_USER variable as follows:

- To set the REMOTE_USER to the value of the SiteMinder log-in user name, set the Web Agent's SetRemoteUser parameter to yes.

  The default for this parameter is no, which leaves the REMOTE_USER variable blank.

  Note: Prior to SiteMinder Web Agent 5.x QMR 2, the SetRemoteUser parameter affected only IIS web servers; Apache and Sun Java System Agents always set REMOTE_USER to the SiteMinder logged-in user name. If you install or upgrade from Agents prior to 5.x QMR 2, note that REMOTE_USER is no longer enabled by default.

- To set the REMOTE_USER variable based on a specific user account instead of the logged-in user's credentials:
  - Enable the SetRemoteUser parameter by setting it to yes.
  - Set the RemoteUserVar parameter. This parameter instructs the Agent to populate the REMOTE_USER variable based on the value from an HTTP-WebAgent-Header-Variable response attribute. Use this to integrate with legacy applications.

    To configure the RemoteUserVar parameter, enter only the name of the response variable. For example, to return an HTTP-WebAgent-Header-Variable such as "user=ajohnson", set the RemoteUserVar parameter to the value user.
  - Bind the header variable to an OnAuthAccept rule. Do not use an existing HTTP header variable response; create a new one.

    Note: For more information, see the Policy Server documentation.
- To revert to the default, which leaves REMOTE_USER blank, return the SetRemoteUser parameter to no.

Note: Be sure to take security consequences into consideration before configuring SetRemoteUser or RemoteUserVar.

SiteMinder Default HTTP Headers

SiteMinder default HTTP headers instruct applications how to collect user data and apply that information to display personalized content for each user.

As part of the Web application environment, the SiteMinder Agent submits default HTTP headers to the web server, and the web server makes them available for Web applications. You can use these headers to include functions and enable your Web applications to personalize content. Headers can store information such as a user's name and the type of action a user is authorized to perform.

The Agent sends these headers regardless of whether or not they are called from a Web application; however, you can disable some of these headers so that they do not use up header space.

The following SiteMinder default HTTP headers are available for Web Agents:

HTTP_SM_AUTHDIRNAME

Indicates the name of the directory against which the Policy Server authenticates the user. The administrator specifies this directory with the Administrative UI.

HTTP_SM_AUTHDIRNAMESPACE

Identifies the directory namespace against which the Policy Server authenticates the user. The administrator specifies this namespace with the Administrative UI.

HTTP_SM_AUTHDIROID

Indicates the directory object identifier (OID) from the Policy Server database.

HTTP_SM_AUTHDIRSERVER

Indicates the directory server against which the Policy Server authenticates the user. The administrator specifies this directory server with the Administrative UI.

HTTP_SM_AUTHREASON

Indicates the code the Web Agent returns to the user after a failed authentication attempt or secondary authentication challenge.

HTTP_SM_AUTHTYPE

Indicates the type of authentication scheme the Policy Server uses to verify the user's identity.

HTTP_SM_DOMINOCN

Identifies the user's Domino canonical name if a Domino LDAP directory is used to authenticate users.

Example: HTTP_SM_DOMINOCN="CN=jsmith/O=netegrity."

HTTP_SM_REALM

Indicates the SiteMinder realm in which the resource exists.

HTTP_SM_REALMOID

Indicates the realm object ID that identifies the realm where the resource exists. This ID is may be used by third party applications to make calls to the Policy Server.

HTTP_SM_SDOMAIN

Indicates the Agent's local cookie domain.

HTTP_SM_SERVERIDENTITYSPEC

Indicates the Policy Server identity ticket that keeps track of the user identity. The Web Agent uses this to access content protected by anonymous authentication schemes so that it can personalize the content for the user.

HTTP_SM_SERVERSESSIONID

Indicates a unique string that identifies a user session.

HTTP_SM_SERVERSESSIONSPEC

Indicates the ticket that contains user session information. Only the Policy Server knows how to decode this information.

HTTP_SM_SESSIONDRIFT

Indicates the amount of time the Web Agent can keep a session active using the information in its cache before validating the session with the Policy Server. The session server at the Policy Server must be enabled and a session validation period must be configured for this header to be set.

HTTP_SM_TIMETOEXPIRE

Indicates the amount of time remaining for a SiteMinder session.

HTTP_SM_TRANSACTIONID

Indicates the agent-generated unique ID for each user request.

HTTP_SM_UNIVERSALID

Identifies the Policy Server-generated universal user ID. This ID is specific to the customer and identifies the user to the application, but it is not the same as the user login.

HTTP_SM_USER

Indicates the login name of the authenticated user. If a user does not provide a user name at log in, such as certificate-based authentication, then this variable is not set.

HTTP_SM_USERDN

Identifies an authenticated user's distinguished name as determined by the Policy Server.

For anonymous authentication schemes, this returns a Globally Unique Identifier (GUID).

HTTP_SM_USERMSG

Identifies the text that the Agent presents to the user after an authentication attempt. Some authentication schemes supply challenge text or a reason why an authentication has failed.

More Information

Disable Default HTTP Header Variables (see page 125)

Disable Default HTTP Header Variables

Many system platforms have an HTTP header limit of 4096 bytes. To avoid exceeding this limit and to allow space for custom response variables, you can disable some of SiteMinder's default HTTP header variables.

The default variables are grouped into the following categories:

Note: You cannot disable individual variables. You can only disable a category of several variables.

- Authentication source variables
  - SM_AUTHDIRNAME
  - SM_AUTHDIRSERVER
  - SM_AUTHDIRNAMESPACE
  - SM_AUTHDIROID
- User Session variables
  - SM_SERVERSESSIONID
  - SM_SERVERSESSIONSPEC
  - SM_SERVERIDENTITYSPEC
  - SM_SESSIONDRIFT
  - SM_TIMETOEXPIRE
- User Name variables
  - SM_USER
  - SM_USERDN
  - SM_DOMINOCN

To disable the default use of HTTP header variables do any of the following tasks:

- To disable authentication source variables, set the value of the DisableAuthSrcVars parameter to yes.
- To disable user session variables, set the value of the DisableSessionVars parameter to yes.
- To disable user name variables, set the value of the DisableUserNameVars parameter to yes.

  Note: If you are using CA Identity Manager, or any application that might use the variables in this category, ensure the value of this parameter is set to no (enabled).

Example Applications that Use SiteMinder Default HTTP Headers

The sections that follow contain examples of different applications that call out the SiteMinder environment headers. They also show how you can use the default headers in applications.

Extract HTTP Headers Using a Shell Script

To access the default HTTP headers, you can extract their values from the environment and display them. This example uses a shell script, called TEST.SH, to return these variables to the browser.

```
#!/bin/sh
# First put out a valid header
echo Content-Type: text/html
echo
# Then echo out the variables
echo HTTP_SM_USER: $HTTP_SM_USER
echo HTTP_SM_AUTHTYPE: $HTTP_SM_AUTHTYPE
echo HTTP_SM_AUTHDIRNAME: $HTTP_SM_AUTHDIRNAME
echo HTTP_SM_SERVERSESSIONID: $HTTP_SM_SERVERSESSIONID
echo HTTP_SM_SESSIONID: $HTTP_SM_SESSIONID
```

Extract HTTP Headers Using NSAPI

This example shows an application that makes an Sun Java System API (NSAPI) call to retrieve a header. The name/value pairs exist in the request structure's header block.

The header is accessed as follows:

```
char * user;
user = pblock_findval("HTTP_SM_USER", rq->headers);
```

Extract HTTP Headers Using PERL

Accessing environmental variables from the Practical Extraction and Report Language (PERL) is a standard practice. The following script is a PERL version of the shell script TEST.SH that returns the variables to the browser:

```
#!/usr/win32/perl117
# Example Test Program To Echo Back Default
# Web Agent Headers from a CGI environment
#
# First put out a valid CGI Header
print “Content-Type: text/html\n\n”;
# Now print out the standard HTML document tags
print “<HTML>”;
print “<HEAD><TITLE> Test Web Agent Headers </TITLE></HEAD>”;
print “<BODY BGCOLOR=#ffffff>”;
print “<H1> Test Web Agent Headers </H1>”;
print “HTTP_SM_USER: $ENV{‘HTTP_SM_USER’} <BR>”;
print “HTTP_SM_AUTHTYPE: $ENV{‘HTTP_SM_AUTHTYPE’} <BR>”;
print “HTTP_SM_SESSIONID: $ENV{‘HTTP_SM_SESSIONID’} <BR>”;
# Now end the HTML file output
print “</BODY>”;
print “</HTML>”;
```

The following PERL script returns all the environment variables to the browser, not just SiteMinder variables.

```
#!/export/home/iplanet/server4/install/perl
print "content-type: text/html\n\n";

print "<HTML>\n";

print "<HEAD>\n";
print "<TITLE>echo cgi env. vars.</TITLE>\n";
print "<H2>Echo CGI Environment Variables</H2>\n";
print "</HEAD>\n";
print "<BODY>\n";
print "<HR>\n";
print "<H3>Environment Variables</H3>\n";
print "<UL>\n";
foreach $key (keys %ENV) {
print "<LI>$key = $ENV{$key}\n";
}
print "</UL>\n";
print "</BODY>\n";
```

Extract HTTP Headers Using ASP

To access the default HTTP headers, you can extract their values from the environment and display them using an ASP script.

To use an ASP script to return SiteMinder variables, your script has to include the ALL_HTTP header, which retrieves all the environment variables, and code that parses out the SiteMinder variables. Use the following script as an example.

```
<HTML>
<HEAD><TITLE> Test Web Agent Headers </TITLE></HEAD>
<BODY BGCOLOR=#ffffff>
<TABLE BORDER=1>
<TR><TD VALIGN=TOP><B>Variable</B></TD><TD VALIGN=TOP><B>Value</B></TD></TR>
<% For Each key in Request.ServerVariables %>
<TR>
<TD><% = key %></TD>
<TD>
<%
if Request.ServerVariables(key) = "" Then
if GetAttribute(key) = "" Then
Response.Write " " ' To force border around table cell
else
Response.Write GetAttribute(key)
end if
else
Response.Write Request.ServerVariables(key)
end if
Response.Write "</TD>"
%>
</TR>
<% Next %>
</TABLE>

<% Function GetAttribute(AttrName)
                    Dim AllAttrs
                    Dim RealAttrName
                    Dim Location
                    Dim Result

                    AllAttrs = Request.ServerVariables("ALL_HTTP")
                    RealAttrName = AttrName

                    Location = instr(AllAttrs, RealAttrName & ":")

                    if Location <= 0 then
                    GetAttribute = ""
                    Exit Function
                    end if
```

```
                    Result = mid(AllAttrs, Location + Len(RealAttrName) + 1)

                    Location = instr(Result, chr(10))
                    if Location <= 0 then Location = len(Result) + 1

                    GetAttribute = left(Result, Location - 1)
End Function %>
```

Header Variables and End-User IP Address Validation

When a SiteMinder Web Agent receives a request that follows an initial request by that same user, the Agent validates the session cookie sent with the subsequent request by comparing the IP address of the requesting user with the IP address encrypted inside the session cookie. The address inside the cookie is generated by the Agent during the user's initial request.

Mechanisms used to balance and manage incoming network traffic, such as firewalls, load balancers, cache devices, and proxies can alter the user's IP address or make it appear as if all incoming requests are coming from a single or small group of IP addresses. As a result, the Web Agent's IP checking becomes ineffective. The Web Agent can now perform IP checking in these network environments using a custom HTTP header and a configurable list of safe proxy IP addresses.

The following table lists the terminology for new IP checking functionality.

Term	Definition
HTTP Request Header	A name/value pair that describes a single element of an HTTP request.
Custom IP Header	A user-defined HTTP request header used by intermediate HTTP network applications or hardware devices to store the requestor's IP address.
IP Checking	Feature that enables the Web Agent to check requests for authenticity by comparing the REMOTE_ADDR in the request with the REMOTE_ADDR value stored in the SMSESSION cookie, after an initial request. This feature is also known as IP validation.
REMOTE_ADDR	web server variable representing the IP address of the HTTP client making a request to the web server. Also known as REMOTE_IP or CLIENT_IP. This differs from the Requestor IP Address when a proxy server, NAT firewall, or other network service or device sits between the requestor and the target web server.

Term	Definition
Requestor	The initiator of an HTTP request, typically a user at a browser.
Requestor IP Address	The IP address of the user making the original HTTP request.
Single Sign-on	Feature that requires a user to enter credentials for secure access to a protected Web site only once during a session.
SMSESSION cookie	HTTP mechanism used by Web Agents to track single sign-on state.

How Custom Headers Validate IP Addresses

The Web Agent can now use a custom HTTP header to determine a user's IP address instead of using the REMOTE_ADDR variable. If a proxy or other device sets a custom client IP header and the Web Agent is configured to look for that header on an incoming request, the Agent uses that header as the source of the client IP information.

In addition to configuring a custom header, you can set up a list of proxy IP addresses. If the REMOTE_ADDR matches an address in the proxy list, the Web Agent retrieves the user's IP address from the custom header. Otherwise, the user's IP address is obtained from the REMOTE_ADDR.

After the Web Agent resolves the requestor's IP address, the address is stored and used for request processing. If an address cannot be resolved, the IP address is set to unknown.

The Web Agent logs where the client IP address was resolved from to facilitate any debugging that may be necessary.

Configure IP Address Validation

IP checking is implemented using two parameters:

- CustomIpHeader—each entry specifies an HTTP header that the Web Agent should look for to find the requestor's IP address. If no value is specified for this parameter, the default is an empty string. No maximum length is enforced and the value may be any string that contains a valid HTTP header value, for example, HTTP_ORIGINAL_IP.

  You can define this setting centrally, in an Agent Config Object or locally, in a configuration file. For central agent configuration, you can designate this parameter as a multi-value setting. For local configuration, each entry should be added on a separate line. For example:

  CustomIpHeader="HTTP_ORIGINAL_IP"

  CustomIpHeader="HTTP_CLIENT_IP"

  Note: The CustomIpHeader parameter can be used without a proxy address list being defined; however, the proxy definition list cannot be used without at least one custom IP header defined.

- ProxyDefinition—each entry specifies the IP address of a proxy (such as a cache device) that requires the use of a custom HTTP header to resolve requestor IP addresses. If no value is specified for this parameter, the default is an empty string.

  No maximum length is enforced for this parameter, and the value may be any string that contains a valid IP address, for example, 111.123.23.11. Do not enter server names or fully-qualified DNS host names.

  You can define this setting centrally, in an Agent Config Object or locally, in a configuration file. For central agent configuration, you can designate this parameter as a multi-value setting. For local configuration, each entry should be added on a separate line. For example:

  ProxyDefinition=111.12.1.1

  ProxyDefinition=112.11.2.2

  Note: These settings are independent of the TransientIPCheck and PersistentIPCheck parameters.

Note: For all traditional Web Agents, these parameters are in the WebAgent.conf file. For framework Agents, these values are in the LocalConfig.conf file.

IP Address Validation with Previous Web Agent Releases

In an environment of 6.x QMR 2 or 3 Web Agents and older Agents, single sign-on may be affected if IP checking is configured.

Web Agents prior to v6.x QMR 2 and 5.x QMR 7 will not be able to resolve the requestor IP address and as a result, SMSESSION cookies created by those Web Agents may be discarded by 6.x QMR 2 or 3 Web Agents. This includes custom Agents using the SDK to generate SMSESSION cookies, Application Server Agents, and any other SiteMinder Agents in a single sign-on environment that use SMSESSION cookies.

Conversely, 6.x QMR 2 and 3 Web Agents may resolve a requestor's IP address, which then differs from the address resolved by an older Agent.

Preserve HTTP Headers

You can configure the Web Agent to save existing HTTP headers instead of replacing them when new headers are created. This feature is useful for applications that generate multiple SiteMinder responses with the same name but with different values, and need to be included in headers. If there are multiple instances of the same HTTP header, the web server handles this by generating a single header with all the relevant header values separated by commas.

By default, the Web Agent does not preserve headers as a precaution against applications using the wrong header values. For Sun Java System, Domino, and Apache Web Agents to preserve HTTP headers, set the PreserveHeaders parameter to yes. The default value is no.

Control How HTTP Header Resources are Cached

The AllowCacheHeaders attribute tells the Web Agent how to handle cache-related request headers. Specifically, this settings tells the Agent whether or not it should remove the if-modified-since or if-none-match request headers before the Agent passes a request to the web server where it is installed. The action taken by the Web Agent affects whether or not a browser uses cached pages.

Note: This attribute does not affect auto-authorized resources. Auto-authorized resources include those matched by the IgnoreExt setting. For these files, the default cache operations are determined by the browser's and web server's own cache settings.

Security Issues Related to Caching HTTP Header Resources

The setting of the AllowCacheHeaders parameter may have security implications.

The setting are as follows:

- Yes—setting AllowCacheHeaders to yes introduces security issues. Pages which are personalized by an application on the web server but do not have the appropriate cache control headers set may become cached in the browser or any HTTP intermediary. This can introduce unexpected behavior and allow a browser to save sensitive data to the disk.

  Additionally, setting this parameter to yes prevents the Web Agent from enforcing session timeouts across all resources. If a resource is served from browser cache, the Agent has no chance to get the SMSESSION cookie and validate the session. Therefore, be sure to evaluate your session security needs before enabling this setting.

- No—the default. If you choose this option, the Web Agent removes the cache-related headers from requests for protected resources. The web server now treats the request as an unconditional request, and will not perform any cache validation operations.

  Note: If you are configuring the LogOffURI parameter, we recommend accepting no as the default value. Otherwise, the browser will deliver a cached version of the LogOffURI resource and the user session will not be terminated.

- None—if you set AllowCacheHeaders to none, the Web Agent tells the web server to remove all cache-related headers for protected and unprotected resources.

If the session has been terminated, the browser will not use what is in cache, regardless of this setting.

Note: See RFC 2616, Section 13 "Caching in HTTP" for more information about HTTP/1.1 caching mechanisms."

Use Lower Case HTTP in Headers (for Sun Java System, Apache, Domino)

If you have server applications that are case-sensitive, you can specify the case of the Agent's HTTP headers. The Web Agent defaults to lower case headers.

For example, Sun Java System web servers, by default, provide the HTTP header variables in lower-case, such as http_sm_user.

Note: IIS Web Agents do not benefit from this feature, because IIS forces all headers to an upper case format.

To use lower case headers, set the LowerCaseHTTP parameter to yes. If you require upper-case header variables, set LowerCaseHTTP to no.

More information:

Record the Transaction ID in Sun Java System Web Server Logs (see page 159)

Set the HTTP Header Encoding Spec

The HTTPHeaderEncodingSpec setting affects the encoding of all HTTP header values and all custom HTTP-COOKIE responses.

Use this parameter to support Web applications expecting localized text in specific encodings. Since cookies pass back and forth between the browser and portal through the HTTP protocol, you should use the RFC-2047 HTTPWrapSpec if your chosen encoding puts characters that HTTP traffic considers illegal into the cookies.

For example, some Shift-JIS characters will cause undesirable results if not further encoded by RFC-2047.

For Kanji characters, you can use SECP932, which is a superset of SHIFT-JIS. Though SHIFT-JIS can be used for most Kanji encoding and decoding, CP932 covers an even larger character set.

When HTTPWrapSpec is used, first the data is encoded according to the HTTPHeaderEncodingSpec, then the data is further encoded following the RFC-2047 specification.

The syntax for the parameter is:

encoding_spec, *wrapping_spec*

The *encoding_spec* is a text string that represents one of the following encoding types: UTF-8, Shift-JIS, EUC-J, or ISO-2022 JP. Specify the encoding type you want the Agent to use.

The *wrapping_spec* is the wrapping specification, which must be RFC-2047. Although this variable is optional, we strongly recommend that you include the wrapping specification because the encoding type you choose may generate byte codes that are not compatible with the HTTP protocol.

This is especially true if you use custom HTTP Cookie responses that contain double-byte encoded data. For example, some Shift-JIS characters cause undesirable results if they are not further encoded by RFC-2047. The wrapping also tells the receiving application the type and nature of the encoding so the application can better interpret the encoded text. For example, you may set the parameter to Shift-JIS,RFC-2047.

When RFC-2047 is used, the Agent first encodes the data based on the chosen encoding specification and then further encodes the data following the RFC-2047 specification.

Note: If you leave the HTTPHeaderEncodingSpec setting blank, the default is UTF-8 with no wrapping.

Disable Conformance to RFC 2047

By default, the Web Agent conforms to RFC 2047. However, you can disable this conformance by setting the ConformToRFC2047 parameter to no.

If this parameter does not exist or is set to yes, the Web Agent conforms to RFC 2047.

Use SM_AGENT_ATTR_USRMSG Response for a Forms Challenge

The SM_AGENTAPI_ATTR_USERMSG response enables developers of custom SiteMinder authentication schemes to return custom text to their client applications, as part of a user challenge or for some other purpose.

Beginning with v5 QMR3, the Web Agent has the ability to convert the text from an SM_AGENTAPI_ATTR_USERMSG response to an SMUSRMSG cookie when performing a forms challenge.

To ensure the SMUSRMSG cookie is removed after the challenge is complete, the FCC consumes the cookie (deletes it from the browser) after a successful POST request, as follows:

- In native SiteMinder mode, the Agent deletes the cookie after a successful login, while redirecting back to the target URL.
- In SiteMinder 4.x compatibility mode, the Agent deletes the cookie after generating the FORMCRED cookie, while redirecting back to the target URL.

Note: The SMUSRMSG cookie will be stored for a period of time in the user's browser, and could possibly be transmitted over non-secure HTTP connections. As a result, sensitive data should be avoided.

Web Agents will URL-encode text that is placed in the SMUSRMSG cookie during a forms challenge, to make it safe for HTTP transmission, eliminating spaces and other harmful characters. The FCC decodes this text before making it available to the environment for use in custom FCC functionality.

Note: URL encoding is not implemented unless the text is placed in the SMUSRMSG cookie.

To implement the new functionality, custom authentication scheme developers must generate custom forms-based authentication schemes. When an Sm_AgentApi_Login() call returns SM_AGENTAPI_CHALLENGE, the Agent challenges the requesting user by redirecting to the authentication scheme URL provided by the response to Sm_AgentApi_IsProtected().

When the Web Agent handles an authentication scheme that uses the HTML Forms authentication scheme template, the Agent looks for a SM_AGENTAPI_ATTR_STATUS_MESSAGE response attribute. If the attribute is found, the Agent generates the appropriate SMUSRMSG cookie, while redirecting to the authentication scheme URL. The FCC may then use this cookie during form generation, if appropriate directives are placed in the desired .FCC source file.

Note: For more information, see the Policy Server documentation.

Enable Legacy Variables for HTTP Headers

You can specify which naming convention the Web Agent uses for HTTP headers with the following parameter:

LegacyVariables

Specifies if the Web Agent uses underscores in HTTP header names. With some web servers (such as the Sun Java System), using the underscore character in the HTTP headers causes problems with some applications.

When this parameter is set to no, the HTTP headers will not have underscores, as shown in the following example:

SM*HeaderName*

When this parameter is set to yes, the HTTP headers will use underscores, as shown in the following example:

SM_*HeaderName*

Default: (traditional agents) Yes

Default: (framework agents) No

To enable legacy variables and have the Web Agent use underscores in the HTTP header names, set value of the LegacyVariables parameter to yes.

Define HTTPS Ports

If you are using an SSL connection to the web server (HTTPS) to keep your requests more secure, you must specify the HTTPS ports with the following parameter:

HttpsPorts

Specifies the secure ports the Web Agent listens on if you are using an SSL connection to the web server. If you specify a value for this parameter, you must include all the ports for all the web servers that serve secure requests. If you do not specify a value, the Web Agent reads the HTTP scheme from the web server's context.

If a server is behind an HTTPS accelerator (which converts HTTPS to HTTP), the requests are treated as SSL connections by your browser.

Default: Empty

Example: 80

Example: (multiple ports) 80,8080,8083

To define your HTTPS ports, set the value of the HttpsPorts parameter to the port numbers that use SSL. Use commas to separate multiple port numbers.

Use the HttpsPorts Parameter on Apache 2.x Servers

To use the HttpsPorts parameter with an Apache 2.x Web server (if you plan to use an SSL accelerator with the Apache Web server, for example) make additional configuration changes to your Web server. These changes are required if you use an SSL accelerator or any intermediate device that changes the value of the HTTP_HOST header.

To use the HttpsPorts Parameter with Apache 2.x servers

1. Open the httpd.conf file of your Apache Web server, and then make the following changes:
   - Change the value of the UseCanonicalName parameter to on.
   - Change the value of the ServerName parameter to the following:

     server_name:port_number
2. Modify the following configuration parameter for your Web Agent:
   - Change the value of the GetPortFromHeaders parameter to yes.

Handle Multiple AuthTrans Functions (Sun Java System only)

AuthTrans functions are directives that initialize the Sun Java System web server. In Sun Java System, the web server executes AuthTrans functions in the order that they are listed in the obj.conf file. The Sun Java System server reads through the AuthTrans functions until it finds a function that returns a REQ_PROCEED command. Once a REQ_PROCEED command executes, no other AuthTrans functions are executed.

By default, SiteMinder is the first AuthTrans function and it returns a REQ_PROCEED. To allow other AuthTrans functions to execute, you need to add the EnableOtherAuthTrans parameter and set the value to yes.

The default value for this parameter is no. To enable multiple AuthTrans functions set the EnableOtherAuthTrans parameter to yes.

By adding this parameter, you permit the SiteMinder Web Agent to exist with other functions.

Be sure the SiteMinder Agent function is the first entry in the obj.conf file for the AuthTrans directive. The entry should read:

```
AuthTrans fn="SiteMinderAgent"
```

Custom Error Handling For Applications

Custom error handling allows you to make error information relevant to your application. To customize applications for users, you can add the HTML text displayed by HTTP 500, HTTP 401, and HTTP 403 error pages or, with the exception of 401 errors, you can redirect the user to a URL that points to a custom error page or application.

You can configure customized handling for the following types of errors:

- Server errors—the Agent uses the ServerErrorFile for error pages that result from HTTP 500 web server errors. These error codes are passed to the custom error pages and include:
  - Problems because the Web Agent cannot read values from required HTTP headers.
  - Advanced authentication cookies cannot be parsed or contain an error status.
  - Connectivity problems between the Web Agent and the Policy Server.
- Access Denied errors—the Agent uses the file specified in the Custom401ErrorFile parameter to display customized text for the standard 401 access denied message. These errors occur when a user does not have the appropriate privileges to access a resource.

  Note: Some web servers may append text of their own to the custom text. Consequently, the response pages for these servers will not be completely customized.
- Require cookies errors—if the RequireCookies parameter is set, the Web Agent sets a cookie during basic authentication. If this cookie is not returned by the browser with the basic credentials, the error page designated by the ReqCookieErrorFile parameter is returned, and the Agent denies the user access to the web server.
- Cross-site scripting errors—the Agent uses the file specified in the CSSErrorFile parameter for error pages that result from HTTP 403 cross-site scripting errors. Cross-site scripting can compromise the security of a Web site.

After you create these HTML files or applications, direct the Web Agent to the custom error pages or URLs.

Note: For an Apache server being used as a proxy or reverse proxy server, the Apache Agent will not return custom SiteMinder error pages, but will return the standard Apache HTTP 500 and 403 error pages.

More Information

Error Codes (see page 389)
Server Error 500 Appears Instead of Custom Error Page (see page 297)

Configure Custom Error Handling

To customize applications for users, you can add the HTML text displayed by HTTP 500, HTTP 401, and HTTP 403 error pages or redirect the user to a URL that points to a custom error page or application.

For HTTP 500 and 403 errors only: If you configure the Agent to redirect the user to a URL, it appends the error code, in the form ?SMError=*error_code*, to the URL. If you add standard HTML error text, you can only specify HTML code between the opening and closing Body tags (<body> </body>); you cannot use a complete custom HTML page or customize any text outside the Body tags.

To direct the Web Agent to the custom error pages or URLs, you specify the path where the text files reside or enter the URL in the Agent configuration file.

Notes for Custom 401 Pages

- Do not set the Custom401errorfile parameter to a URL.
- If a value (usable or not) for Custom401errorfile exists, the Agent will check every 60 seconds to see if the file has changed. However, the response is intended to be static in nature. You cannot, for example, insert a "*user_name* denied" type of dynamic message.

  Because re-checking is triggered by the existence of the Custom401errorfile value rather than its usability, you can correct an error without restarting the agent. The correction will be picked up on the next check.
- The customized message file text will not be exposed by other errors. The file pathname will be logged at startup and in the case of error.
- The extent of customization may be limited by the web server, which may add text of its own to the response.
- The size of the customized text file is limited only by the system file size limit.

How to Set Up Error Handling

To direct the Web Agent to a server error, access denied, require cookie, or cross-site scripting (CSS) custom error file or page, specify the file path or URL in the ServerErrorFile, Custom401ErrorFile, ReqCookieErrorFile, or CSSErrorFile parameters respectively. The error files can reside anywhere in your application.

Important! Any URL you configure as a custom error page should be an unprotected resource. To prevent a misconfiguration in which it is not possible to display the specified error file because it is protected, any URL you configure as an error page is added to the ignoreURL list.

Note: If your applications require URLs that include HTML tags, you can encode these characters so the Web Agent does not block a request. For information about encoding HTML characters, see: http://www.cert.org/tech_tips/.

The following examples show a file path and a URL to an error file. The syntax in the example is for a local Agent configuration file. These parameters can also be set in an Agent Configuration Object.

File Path:

CSSErrorFile="C:\error\error.txt"

ReqCookieErrorFile="C:\custompages\error.txt"

ServerErrorFile="C:\error\error.txt"

Custom401ErrorFile="C:\error\accessdenied.txt"

URL:

CSSErrorFile="http://www.mycompany.com.error.jsp"

ReqCookieErrorFile="http://www.myorg.com.error.asp"

ServerErrorFile="http://www.mycompany.com.error.jsp"

More Information

Custom Error Handling For Applications (see page 139)

Chapter 10: Manage User Access with IIS

This section contains the following topics:

Store Encrypted Credentials in a Page File (IIS 5.0 Only)

If you have Web Agents running on IIS 5.0 web servers, you can enforce the native IIS security mechanisms by having the Web Agent provide a Windows user security context. Use the following parameter:

InsecureServer

Allows Web Agents for IIS 5 to enforce native IIS security mechanisms by providing a Windows user security context. Add this parameter to the Agent Configuration Object or local configuration file with the value you want.

If this parameter is set to yes, the Web Agent stores encrypted credentials in paged memory, which can be written to the operating system's page file and saved to a hard disk.

Important! If your hard disk is stolen or compromised, confidential data could be exposed.

If this parameter is no, the Web Agent stores encrypted credentials in protected kernel memory. This setting is more secure, but it places more demands on the physical memory of your IIS server.

Default: No

Note: This parameter applies to IIS 5.0 Web Agents only. It is not used in Framework Web Agents.

To store encrypted user credentials in paged memory, add the InsecureServer parameter to the Agent's Configuration Object (or file), and set this parameter to yes.

Use an IIS Proxy User Account (IIS Only)

If users try to access resources on an IIS web server protected by SiteMinder, the Web Agent may deny access if those users lack sufficient IIS privileges for those resources. For example, if users are stored in an LDAP user directory on a UNIX system, those users may not have access to the Windows system with the IIS web server.

The IIS web server has a default proxy account that has sufficient privileges for users who are granted access by SiteMinder. The Web Agent uses the values of the DefaultUserName and DefaultPassword parameters as credentials even if the user has a valid Windows security context.

To configure the IIS Web Agent to use a proxy user account

1. Set the value of the ForceIISProxyUser parameter to *one* of the following values:
   - If access to the applications on the IIS server is based on the users' credentials themselves, set the value of the ForceIISProxyUser parameter to yes.
   - If access to the applications on the IIS server is based on a specific account (such as a proxy) which acts on behalf of the users, set the value of the ForceIISProxyUser parameter to no.
2. If you are *not* using either of the following Windows features, continue with Step 3:
   - The Windows authentication scheme
   - The Windows User Security Context
3. Enter the user name for the proxy user account in the DefaultUserName parameter. If you are using a domain account, and the local machine is not a part of that domain, use the syntax shown in the following example:

   DefaultUserName=*Windows_domain\acct_with_admin_privilege*

   Otherwise, specify just the user name.
4. Enter the password associated with the existing Windows user account in the DefaultPassword parameter.

   Important! We recommend setting this parameter in your Agent Configuration Object because you can encrypt it. If you set it in a local configuration file, the value is stored unencrypted in plain text.

   The IIS Proxy account is configured.

Enable Anonymous User Access

If you do not want users to have access as the proxy user, you can set the following parameter:

UseAnonAccess

Instructs the IIS Web Agent to execute the web application as an anonymous user, instead of using credentials of the proxy user.

Default: No

Note: This parameter applies to IIS Web Agents only.

To enable anonymous user access, set the UseAnonAccess parameter to yes.

Use the NetBIOS Name or UPN for IIS Authentication

In an IIS network, you may have a NetBIOS name that is different than the domain name for the location of a requested resource. When a user tries to access a protected resource and there are multiple domain controllers, user authentication fails and the web server log shows an "IIS logon failure." You can control whether the UPN or NetBIOS name is sent to the IIS web server with the following parameter:

UseNetBIOSforIISAuth

Specifies whether the IIS 6.0 Web Agent sends the user principal name (UPN) or the NetBIOS name to the IIS 6.0 web server for IIS user authentication.

Note: This parameter is valid only if an Active Directory user store is associated with the Policy Server.

If you enable this parameter, the Policy Server extracts the UserDN, the UPN, and the NetBIOS name from the Active Directory during SiteMinder authentication, and sends this data back to the IIS 6.0 Web Agent.

Depending on whether or not you selected the Run in Authenticated User's Security Context option for the user directory with the Administrative UI and how you set the UseNetBIOSforIIAuth parameter, a user's logon credentials are sent as follows:

- When the UseNetBIOSforIISAuth parameter is set to no, the IIS 6.0 Web Agent sends the UPN name.
- When the UseNetBIOSforIISAuth parameter is set to yes, the Web Agent sends the NetBIOS name.

The IIS web server authenticates the user with the credentials it receives from the Web Agent.

Default: No

To have the Web Agent use the NetBIOS name for IIS authentication, set the UseNetBIOSAuth parameter to yes.

How to Configure the NT Challenge/Response Authentication (IIS Only)

The IIS Web Agent supports the NT challenge/response authentication scheme. With NT challenge/response authentication, the IIS web server challenges the user's Internet Explorer browser when a user requests access to a resource. The challenge is a mathematical calculation based on the user's password that is stored on the user's client system. The browser returns the results of the calculation to the web server, which compares the response with the password information in its database and does the same calculation. If the results match, the server allows the user access. This process is transparent to the user.

Note: The NT challenge/response authentication scheme only works with Internet Explorer browsers.

There are two ways you can choose to implement the challenge/response authentication scheme:

- Challenge users when they try to access a protected resource (in single sign-on environments, users are only challenged the first time they request a resource)
- Have your users configure the automatic logon feature of their Internet Explorer browser.

  The automatic logon feature allows users to access a resource without being challenged. The authentication process still takes place, but the NT challenge/response process between the browser and the server is transparent to the user. Automatic logon is typically used for intranets where security is less strict and you want users to have seamless access to resources. Automatic logon is not recommended for communication across the Internet and is usually not possible because the user's Windows account would need to be in the same Windows domain system as the web server.

SiteMinder Agents use credential collectors to gather a user's Windows credentials for the NT challenge/response authentication scheme. The agent supports the extension .NTC for collecting NTLM credentials.

Note: NTCEXT only has to be set if the you wish to change this default behavior.

To make SiteMinder operate with NT challenge/response authentication, use the following process:

1. Set up the NT Challenge response authentication for the IIS web server with the following tasks:
   a. Map the .ntc file extension.
   b. Create and configure the virtual directory, and then ensure that it requires the NT challenge and response credentials.

2. Configure the NT challenge/response authentication scheme in the Administrative UI.
3. Specify an NTLM credential collector.
4. Configure policies for NT Challenge/Response authentication using the Administrative UI.
5. (Optional) Have your users configure the automatic logon feature of their Internet Explorer browser.

   The NT Challenge Response Authentication for IIS is configured.

More Information

Configure Automatic Logon for Internet Explorer (see page 150)
Configure the Challenge/Response Authentication Scheme (see page 151)
Specify an NTLM Credential Collector (see page 152)

Map the .NTC File Extension

You must map the .NTC file extension to the ISAPIWebAgent.dll application to configure tNT Challenge/Response Authentication on the IIS Web Server.

To map the .NTC file extension

1. Open the Internet Services Manager.
2. Right-click Web Sites in the left pane, and then right click Default Web Site in the right pane and select Properties.

   The Default Web Site Properties dialog appears.
3. Click the Home Directory tab.
4. In the Application Settings group box, click Configuration.

   The Application Configuration dialog appears.
5. Click Add.

   The Add/Edit Application Extension Mapping dialog opens.

   a. In the Executable field, click Browse and locate the following file: *web_agent_home*/bin/ISAPIWebAgent.dll.

   b. Click Open.

   c. In the Extension field, enter .ntc.
6. Click OK three times.

   The Add/Edit Application Extension Mapping dialog, the Application Configuration dialog and the Default Web Site Properties dialog close. The .ntc file extension is mapped.

Create and Configure the Virtual Directory

After mapping the .ntc file extension, you must create and configure a virtual directory. The virtual directory must require NT challenge and response for credentials.

To configure the virtual directory and ensure it requires NT challenge and response for credentials

1. Create the following virtual directory on your IIS web server:

   /siteminderagent/ntlm

2. Open the Internet Services Manager.
3. Click the Web Sites folder in the left pane.
4. Right-click Default Web Site in the right pane, and select Properties.
5. The Default Web Site Properties dialog appears.
6. Click the Directory Security tab.
7. In the Anonymous Access and Authentication Control group box, click Edit.

   The Authentication Methods dialog appears.

8. Do the following:

   a. Clear the Enable Anonymous Access check box.

   b. Select the Integrated Windows Authentication check box.

9. Click OK twice.

   The Authentication Methods dialog and the Default Web Site Properties dialog close. The virtual directory is configured and requires NT challenge and response for credentials.

Note: You may need to reboot and then verify that the virtual directory and its appropriate settings are correct.

Configure Automatic Logon for Internet Explorer

If you want to authenticate users *without* having the Web Agent challenge them for their credentials, have each user configure the NT automatic logon feature of the Internet Explorer browser by changing the security settings.

To configure automatic logon

1. Start the Internet Explorer browser.
2. Click *one* of the following:
   - The View menu (on Internet Explorer 4.x or 5.x)
   - The Tools menu (on Internet Explorer 6.0)
3. Select Internet Options.

   The Internet Options dialog opens.
4. Click the Security tab.
5. Click the correct security zone (internet, intranet, trusted, or restricted).
6. Click one of the following:
   - The Custom radio button (on Internet Explorer 4.x).
   - The Custom Level radio button (on Internet Explorer 5.x or 6.0).
7. Click Settings.
8. Scroll down to the User Authentication section. Under the Logon option, click the Automatic Logon with current username and password radio button.
9. Click OK twice.

   The Security Settings dialog and the Internet Options dialog close. Your settings are saved, and automatic login is configured.

Configure the Challenge/Response Authentication Scheme

To implement NT Challenge/Response authentication, you must create an authentication scheme using the Administrative UI.

Note: For more information, see the Policy Server documentation.

Use the following settings when creating an NT Challenge/Response authentication scheme:

- In the Scheme Common Setup group box:

  Authentication Scheme Type: Windows Template

- In the Scheme Type Setup tab:

  Server Name: enter the fully qualified domain name, for example:

  server1.myorg.com

  Target: /siteminderagent/ntlm/smntlm.ntc

  Note: The directory should correspond to the virtual directory already set up by the installation. The target file, smntlm.ntc, does not need to exist and can be any name that ends in .ntc or the custom MIME type that you use in place of the default.

- In the Advanced tab:

  Library: smauthntlm

More Information

MIME Types for Credential Collectors (see page 232)

Specify an NTLM Credential Collector

The NTLM credential collector (NTC) is an application within the Web Agent. It collects NT credentials for resources that are protected by the Windows authentication scheme. This scheme applies to resources on an IIS web server accessed by Internet Explorer browsers.

Each credential collector has an associated MIME type. For IIS, the NTC MIME TYPE is defined in the following parameter:

NTCExt

Specifies the MIME type associated with the NTLM credential collector. This collector gathers NT credentials for resources that are protected by the Windows authentication scheme. This scheme applies to resources on IIS web servers that are accessed by the Internet Explorer browser.

You can have multiple extensions in this parameter. If you are using an Agent Configuration Object, select the multi-value option. If you are using a local configuration file, separate each extension with a comma.

Default: .ntc

If your environment already uses the default extension specified by the previous parameter, you can specify a different MIME type.

To change the extension that triggers the credential collector, add a different file extension to the NTCExt parameter.

More Information

Use Credential Collectors for Authentication and Single Sign-On (see page 231)

How to Implement an Information Card Authentication Scheme

CA SiteMinder supports an Information Card Authentication Scheme (ICAS) that implements Windows CardSpace. Users who request access to protected resources can select an authentication card. SiteMinder uses the information contained in the card to verify the user's identity.

Implementing an ICAS requires configuration changes on the following SiteMinder components:

- The server hosting the SiteMinder Web Agent
- The SiteMinder Policy Server
- The smkey database

Use the following process:

1. Do the following tasks on the web server:
   a. Enable SSL communication on the IIS 6.0 web server.

      Note: For more information, see your Microsoft documentation, or go to http://support.microsoft.com/

   b. Export the web server certificate as a .pfx file.
   c. Customize the SiteMinder InfoCard.fcc template.
2. Do the following tasks on the Policy Server:
   a. Install the JCE on the policy server
   b. Update the java.security file on the policy server.
   c. Update the config.properties file on the Policy Server.
   d. If you do not already have an smkey database, Create one with the Policy Server Configuration wizard.
   e. Add the .pfx file certificate from the web server to the smkey database.
   f. Configure the user directory in the Policy Server
   g. Create a custom authentication scheme for CardSpace using the Administrative UI
   h. (Optional) Store the claims in the session store to use in responses.
   i. (Optional) Enable personalization by allowing the retrieval of claim values from the session store.
   j. (Optional) Configure an active response to retrieve a stored claim value.

Export the Web Server Certificate to your smkey Database

Part of configuring an ICAS involves exporting the SSL certificate from the web server which hosts your SiteMinder Web Agent in a different format and placing it in your smkey database.

To export the web server certificate

1. Open the Internet Information Services (IIS) Manager.
2. Expand the web server, and then expand the web sites folder.
3. Right-click the Default Web Site (or the one on which you want to configure the ICAS) and select Properties.

   The Properties page appears.
4. Click the Directory Security tab.
5. In the Secure Communications pane, click Server Certificate.

   The Web Server Certificate Wizard opens.
6. Export the certificate to a .pfx file using the wizard.

   Important! Remember the password you choose during this process. You will need it in the future.
7. Click Finish to close the wizard, and then click OK to close the Properties page.
8. Close the IIS Manager.
9. Deliver the .pfx certificate file to your Policy Server administrator and have the administrator import it into the smkey database.

   Note: For more information, see the *Federation Security Services Guide*.

Configure an FCC Template for an Information Card Authentication Scheme

The SiteMinder Web Agent includes a Forms Credential Collector (FCC) template that you can use to implement an ICAS in SiteMinder.

To configure the FCC template for an Information Card Authentication Scheme

1. Open the following default FCC file with a text editor:

   web_agent_home\samples_default\forms\InfoCard.fcc

2. Save a *copy* of the file to the following directory (this preserves the default FCC settings in case you need them later):

   web_agent_home\samples\forms\

3. Record the following information from your copy of the FCC file:

   Important! The Policy Server needs this information for its configuration.

   - The fully-qualified domain name of the IIS Web server that hosts the Web Agent.
   - The name of the FCC file you saved in Step 2.
   - The value (*without* quotation marks) of the requiredClaims parameter tag in the FCC file from Step 2. See the following example:

     http://schemas.xmlsoap.org/ws/2005/05/identity/claims/privatepersonalidentifier

4. (Optional) Use the text editor to make any of the following changes in copy of the FCC file.

   - To use a custom logo, replace the netegrity_logo.gif file with your own graphic and update the following link in the FCC file accordingly:

     <img alt="Logo" src="/siteminderagent/dmspages/netegrity_logo.gif">

Chapter 11: Track User Activities

This section contains the following topics:

Track User Activities or Application Usage with Auditing

You can you can measure how often applications on your web site are used, or track user activities with auditing. Auditing is controlled with the following parameter:

EnableAuditing

Specifies whether the Web Agent logs all successful authorizations that are stored in the user session cache. When enabled, user authorizations are logged even when the Web Agent uses information from its cache instead of contacting the Policy Server. Web Agents log user names and access information in native web server log files when users access resources.

Default: No

Both the Policy Server and Web Agent audit user activity. The Web Agent sends a message to the Accounting service each time a user is authorized from cache to access resources. This action ensures that the Accounting service is tracking successful authorizations for the Web Agent and the Policy Server. If the Web Agent cannot successfully send an audit message to the Accounting service for an authorization, access to the resource is denied. You can then run a SiteMinder activity report from the Administrative UI. The reports from the Policy Server show user activity for each SiteMinder session.

Note: For more information, see the Policy Server documentation.

To track user activity or application usage with auditing, set the value of the EnableAuditing parameter to yes.

Transaction IDs

The Web Agent generates a unique transaction ID for each successful user authorization request. The Agent adds the ID to the HTTP header. The ID is also recorded in the following logs:

- Audit log
- Web server log (if the server is configured to log query strings)
- Policy Server log

You can track user activities for a given application using the transaction ID.

Note: For more information, see the Policy Server documentation.

The transaction ID appears in the log as a mock query parameter in the log that is appended to the end of an existing query string. The following example shows transaction ID (in bold) appended to a query string (which ends with STATE=MA):

172.24.12.1, user1, 2/11/00, 15:30:10, W3SVC, MYSERVER, 192.168.100.100, 26844, 47, 101, 400, 123, GET, /realm/index.html, STATE=MA&**SMTRANSACTIONID=0c01a8c0-01f0-38a47152-01ad-02714ae1**

If no query parameters are in the URL, the Agent adds the transaction ID at the end of the web server log entry. For example:

172.24.12.1, user1, 2/11/00, 15:30:10, W3SVC, MYSERVER, 192.168.100.100, 26844, 47, 101, 400, 123, GET, /realma/index.html, **SMTRANSACTIONID=0c01a8c0-01f0-38a47152-01ad-02714ae1.**

Note: Web Agents log user names and access information in native web server log files when users access resources.

Record the Transaction ID in Sun Java System Web Server Logs

Valid on Solaris

You can record the SiteMinder transaction ID in the Sun Java System web server logs.

To record the SiteMinder transaction ID in the Sun Java System web server logs

1. Open the magnus.conf file.
2. Add the the following header variable to the existing list of HTTP server variables that you want to log when the web server initializes:

   %Req->headers.SM_TRANSACTIONID%"

   Note: You must enter the header variable in uppercase unless the value of the LowerCaseHTTP parameter is set to yes in your Agent Configuration Object or local configuration file.

   The following example shows the SMTRANSACTIONID header variable in bold at the end of an existing entry. However, you can place it anywhere in the list of variables.

   Init fn="flex-init" access="D:/iPlanet/server4/https-orion/logs/access" format.access="%Ses->client.ip% - %Req->vars.auth-user% [%SYSDATE%] \" %Req->srvhdrs.clf-status% %Req-srvhdrs.content-length% **%Req->headers.- SM_TRANSACTIONID%"**
3. Restart the Sun Java System Server to apply the change.

   The transaction ID appears in the Sun Java System web server logs. The following example shows a web server log entry with the transaction ID in bold:

   11.22.33.44 - user1 [21/Nov/2003:16:12:24 -0500] "GET /Anon/index.html HTTP/1.0" 200 748 **3890b4b9-58f8-4a74df53-07f6-0002df88**

More information:

Use Lower Case HTTP in Headers (for Sun Java System, Apache, Domino) (see page 134)

Record the Transaction ID in Apache Web Server Logs

You can record the SiteMinder transaction ID in the Apache web server logs SMTRANSACTIONID header variable.

To record the SiteMinder transaction ID in the Apache web server logs

1. Open the httpd.conf file.
2. Add the SM_TRANSACTIONID header variable to the LogFormat directive.

   For example:

   ```
   LogFormat "%h %l %u %t \"%r\" %>s %b \"%{SM_TRANSACTIONID}i\"" common
   ```

   Note: For more information about the httpd.conf file and the LogFormat directive, see your Apache web server documentation.
3. Restart the server to apply the change.

   The transaction ID will be recorded in the Apache web server logs.

Record the User Name and Transaction ID in IIS 6.0 Server Logs

Web Agents protecting resources on IIS 6.0 servers do *not* record the SiteMinder transaction ID and authenticated user name in the IIS server logs by default. Because these Web Agents function as ISAPI extensions, the server would have have already logged a transaction. You can force the Web Agent to add this information with the following parameters:

AppendIISServerLog

Instructs the Web Agent to add the authenticated user name and SiteMinder transaction ID to the IIS server log on a separate line.

Default: No

SetRemoteUser

Specifies a value for the REMOTE_USER variable that some legacy applications may require.

Default: No

To record the transaction ID and user name in the IIS server log

1. Set the value of the AppendIISServerLog parameter to yes.
2. Set the value of the SetRemoteUser parameter to yes.

   The IIS server logs will contain the user name and transaction ID.

Chapter 12: Configure Logs

This section contains the following topics:

Logs of Start-up Events

To assist in debugging, startup events are recorded in a log. Each message may provide clues about the problem. These logs are stored in the following locations:

- On Windows systems, these events are recorded in the Windows Application Event log.
- On UNIX systems, these events are sent to STDERR. Apache servers map STDERR to the Apache error_log file, so these events are also recorded in that log.

Error Logs and Trace Logs

You can use the Web Agent logging function to monitor the performance of the Web Agent and its communication with the Policy Server. The logging feature provides accurate and comprehensive information about the operation of SiteMinder processes to analyze performance and troubleshoot issues.

A log is a record of events that occur during program execution. A log consists of a series of log messages, each one describing some event that occurred during program execution. Log messages are written to log files.

Note: IIS 6.0 Web Agents create log files only after the first user request is submitted. Apache 2.0 Web Agents create log files when the Apache server starts.

The Web Agent uses the following log files:

Error log

Contains program and operational-level errors. One example is when the Web Agent cannot communicate with Policy Server. The level of detail output in this log cannot be customized. Error logs contain the following types of messages:

Error messages

Contain program-level errors, which indicate incorrect or abnormal program behavior, or an inability to function as expected due to some external problem, such as a network failure. There are also operational-level errors. This type of error is a failure that prevents the operation from succeeding, such as opening a file or authenticating a user.

Informational messages

Contain messages for the user or administrator that some event has occurred; that is, that a server has started or stopped, or that some action has been taken.

Warning messages

Contain warnings for the user or administrator of some condition or event that is unusual or indicative of a potential problem. This does not necessarily mean there is anything wrong.

Trace log

Contains detailed warning and informational messages, which you can configure. Examples include trace messages and flow state messages. This file also includes data such as header details and cookie variables. Trace logs contain the following messages:

Trace messages

Provide detailed information about program operation for tracing and/or debugging purposes. Trace messages are ordinarily turned off during normal operation. In contrast to informational, warning, and error messages, trace messages are embedded in the source code and can not easily be localized. Moreover, trace messages may include significant data in addition to the message itself; for example, the name of the current user or realm.

You specify the location of both the error and trace log files when you configure the Web Agent. Use the error and trace logs to help solve any issues that may prevent the Web Agent from operating properly.

Note: For Agents on Windows platforms, set the EnableWebAgent parameter to yes to ensure that the Web Agent log gets created. If you leave EnableWebAgent set to no (the default) and set the logging parameters, the Agent log gets created only for Agents on UNIX platforms.

More Information

Set Up and Enable Error Logging (see page 164)
Configure Trace Logging (see page 168)

Parameter Values Shown in Log Files

Web Agents list configuration parameters and their values in the Web Agent error log file, but there are differences between the ways that Traditional and Framework agents do this.

Framework agents record the configuration parameters and their values in the log file exactly as you entered them in the Agent Configuration Object or the local configuration file. All of the parameters, including those which may contain an incorrect value, are recorded in the log file.

Traditional agents process the parameter values before recording them. If the parameter has a proper value, the parameter and its value are recorded in the log file. Parameters with incorrect values are *not* recorded in the log file.

Set Up and Enable Error Logging

The error log is not available until you enable logging and specify a location for the log file. The parameters that enable error logging and determine options such as appending log data are defined in a local configuration file or an Agent Configuration Object at the Policy Server.

Web Agents installed on an IIS 6.0 or Apache 2.0 web server do not support dynamic configuration of log parameters set locally in a local configuration file. Consequently, when you modify a parameter, the change does not take effect until the Agent is restarted. However, these log settings can be stored and updated dynamically if you configure them in an Agent Configuration Object at the Policy Server.

Note: IIS 6.0 Web Agents create log files only after the first user request is submitted. Apache 2.0 Web Agents create log files when the Apache server starts.

To set up and enable error logging

1. If you do not have a log file already, create a new log file and any related directories.
2. Set the value of the LogFile parameter to yes.

   Note: Setting the value of this parameter to yes in a local configuration file of a web server overrides any of the logging settings defined on the Policy Server. For example, when the value of this parameter is set to yes in a LocalConfig.conf file, then log files are generated even if the value of the AllowLocalConfig parameter in the corresponding Agent Configuration object on the Policy Server is set to no. Set the related logging parameters (that define the file name, size, and others) in the LocalConfig.conf file too to override any Policy Server log settings.
3. Specify the full path to the error file, including the file name, in the LogFileName parameter. For example:

   /export/iPlanet/servers/https-jsmith/logs/WebAgent.log
4. (Optional) Set the following parameters (in the Agent Configuration Object on the Policy Server or in the local configuration file):

   LogAppend

   Adds new log information to the end of an existing log file. When this parameter is set to no, the entire log file is rewritten each time logging is invoked.

   Default: No

   LogFileSize

Specifies the size limit of the log file in megabytes. When the current log file reaches this limit, a new log file is created. The new log file uses one of the following naming conventions:

- For framework agents, the new log file has a sequence number appended to the original name. For example, a log file named myfile.log is renamed to myfile.log.1 when the size limit is reached.
- For traditional agents, the new log files are named by appending the date and timestamp to the original name. For example, a log file named myfile.log, is renamed to myfile.log.09-18-2003-16-07-07 when the size limit is reached.

You must archive or remove the old files manually.

Note: Rolling logs are *not* supported for Apache 1.x web servers on UNIX systems. Accept the default or leave this setting blank.

Default: 0 (no rollover)

Example: 80

LogLocalTime

Specifies whether the logs use Greenwich Mean Time (GMT) or local time. To use GMT, change this setting to no. If this parameter does not exist, the default setting is used.

Default: Yes

If you use a local configuration file, your settings will resemble the following example:

```
LogFile="yes"
LogFileName="/export/iPlanet/servers/https-myserver/logs/errors.log"
LogAppend="no"
LogFileSize="80"
LogLocalTime="yes"
```

Error logging is enabled.

Enable Transport Layer Interface (TLI) Logging

When you want to examine the connections between the Web Agent and the Policy Server, enable transport layer interface logging.

To enable TLI logging

1. Add the following environment variable to your web server.

   SM_TLI_LOG_FILE

2. Specify a directory and log file name for the value of the variable, as shown in the following example:

   directory_name/*log_file_name*.log

3. Ensure your Web Agent is enabled.
4. Restart your web server.

   TLI logging is enabled.

Limit the Number of Log Files Saved

You can limit the number of log files that a Web Agent keeps. For example, if you want to save disk space on the system that stores your Web Agent Logs, you can limit the number of log files using the following parameter:

LogFilesToKeep

Specifies the number of Web Agent log files that are kept. New log files are created in the following situations:

- When the Web Agent starts.
- When the size limit of the log file (specified by the value of the LogFileSize parameter) is reached.

Changing the value of this parameter does *not* automatically delete any existing logs files which exceed the number that you want to keep. For example, If your system has 500 log files stored, and you decide to keep only 50 of those files, the Web Agent does *not* delete the other 450 files.

Setting the value of this parameter to zero retains all the log files.

Default: 0

To limit the number of log files saved

1. Archive or delete any existing log files from your system.
2. Set the value of the LogAppend parameter to no.
3. Change the value of the LogFilesToKeep parameter to the number of log files that you want to keep.

How to Set Up Trace Logging

To set up trace logging, use the following process:

1. Set up and Enable Trace logging.
2. Determine what you want to record in the trace log by reviewing the following lists:
   - Trace Log Components and Subcomponents
   - Trace Message Data Fields
   - Data Field Filters
3. Duplicate the default Trace Configuration File.
4. Modify the duplicate file to include the items you want to record.
5. Restart the Web Agent.

Configure Trace Logging

Before you can use trace logging, you must configure it by specifying a name, location, and parameters for the trace log file. These settings control the size and format of the file itself. After trace logging is configured, you determine the content of the trace log file separately. This lets you change the types of information contained in your trace log at any time, without changing the parameters of the trace log file itself.

To configure trace logging

1. Locate the WebAgentTrace.conf file on your web server. Create a duplicate of the file.
2. Open your Agent Configuration Object or local configuration file.
3. Set the TraceFile parameter to yes.

   Note: Setting the value of this parameter to yes in a local configuration file of a web server overrides any of the logging settings defined on the Policy Server. For example, when the value of this parameter is set to yes in a LocalConfig.conf file, then log files are generated even if the value of the AllowLocalConfig parameter in the corresponding Agent Configuration object on the Policy Server is set to no. Set the related logging parameters (that define the file name, size, and others) in the LocalConfig.conf file too to override any Policy Server log settings.
4. Specify the full path to the trace log file in the TraceFileName parameter. This is the file that contains the log output.
5. Specify the full path to the duplicate copy WebAgentTrace.conf file (you created in Step 1) in the TraceConfigFile parameter The trace configuration file that determines which components, subcomponents and data fields to record in the trace log file.

   Note: This file is not used until the web server is restarted.
6. Define the format the trace log file by setting the following parameters in your Agent Configuration Object or local configuration file:

   TraceAppend

   Adds new logging information to the end of an existing log file instead of rewriting the entire file each time logging is invoked.

   Default: No

   TraceFormat

   Specifies how the trace.conf file displays the messages. Choose *one* of the following options:

   - default—uses square brackets [] to enclose the fields.
   - fixed—uses fields with a fixed width.
   - delim—uses a character of your choice to delimit the fields.

- xml—uses XML-like tags. A DTD or style sheet is *not* provided with the Web Agent.

Default: default (square brackets)

TraceDelimiter

Specifies the custom character that separates the fields in the trace.conf file.

Default: No default

Example: |

TraceFileSize

Specifies (in megabytes) the maximum size of a trace file. The Web Agent creates a new file when this limit is reached.

Note: This feature is not supported for Apache 1.x and Sun Java System on UNIX systems. Use the default or leave this setting blank.

Default: 0 (a new log file is not created)

Example: 20 (MB)

LogLocalTime

Specifies whether the logs use Greenwich Mean Time (GMT) or local time. To use GMT, change this setting to no. If this parameter does not exist, the default setting is used.

Default: Yes

7. If you modified the TraceConfigFile parameter, restart the web server so the Web Agent uses the new trace configuration file.
8. Edit the WebAgentTrace.conf file to have Web Agent monitor the activities you want.

   Framework Web Agents do not support dynamic configuration of log parameters set locally in the Agent configuration file. Consequently, when you modify a parameter, the change does not take effect until you restart the web server. However, these log settings can be stored and updated dynamically if you configure them in an Agent configuration object on the Policy Server.

   Note: IIS 6.0 Web Agents create log files only after the first user request is submitted. Apache 2.0 Web Agents create log files when the Apache server starts.

Trace Log Components and Subcomponents

The Web Agent can monitor specific SiteMinder components. When you monitor a component, all of the events for that component are recorded in the trace log. Each component has one or more subcomponents that the Web Agent can also monitor. If you do not want the Web Agent to record all of the events for a component, you can specify only those subcomponents you want to monitor instead.

For example, if you want to record only the single sign-on messages for a Web Agent on a web server, you would specify The Web Agent component and the SSO subcomponent.

The following components and subcomponents are available:

AgentFramework

Records all Agent framework messages. (Applies only to framework agents.) The following subcomponents are available:

- Administration
- Filter
- HighLevelAgent
- LowLevelAgent
- LowLevelAgentWP

AffiliateAgent

Records web Agent messages related to the 4.x Affiliate Agent, which is part of Federation Security Services, a separately-purchased product. (Applies only to framework agents.) The following subcomponent is available:

- RequestProcessing

SAMLAgent

Web Agent messages related to the SAML Affiliate Agent. (Applies only to framework agents.) The following subcomponent is available:

- RequestProcessing

WebAgent

Records all Web Agent log messages. Applies to all Agents *except* IIS 6.0 or Apache 2.0 Agents. The following subcomponents are available:

- AgentCore
- Cache
- authentication
- Responses
- Management

- SSO
- Filter

Agent_Functions

Records all Agent API messages. The following subcomponents are available:

- Init
- UnInit
- IsProtected
- Login
- ChangePassword
- Validate
- Logout
- Authorize
- Audit
- FreeAttributes
- UpdateAttributes
- GetSessionVariables
- SetSessionVariables
- DeleteSessionVariables
- Tunnel
- GetConfig
- DoManagement

Agent_Connection_Manager

Records messages related to internal processing of the Agent API. The following subcomponents are available:

- RequestHandler
- Cluster
- Server
- WaitQueue
- Management
- Statistics

For an explanation of each subcomponent, see the WebAgentTrace.conf file.

Trace Message Data Fields

You can define what each trace message for a specific component contains by specifying which data fields to include in the message.

Data fields use the following syntax:

data:*data_field1*,*data_field2*,*data_field3*

Some data fields are shown in the following example:

data:message,date,time,user,agentname,IPAddr

There may not be data for fields in each message, so blank fields my occur. For example, if you select RealmOID as a data field, some trace messages will display the realm's OID while others will not.

The following data fields are available:

Message

Includes the actual trace message

SrcFile

Includes the source file and line number of the trace message

Pid

Includes the process ID

Tid

Includes the thread ID

Date

Includes the date

Time

Includes the time

PreciseTime

Includes the time, including milliseconds

Function

Includes the function in the code containing the trace message

User

Includes the name of the user

Domain

Includes the SiteMinder domain

Realm

Includes the SiteMinder realm

AgentName

Includes the Agent name being used

TransactionID

Includes the transaction ID

DomainOID

Includes the SiteMinder domain OID

IPAddr

Includes the client IP address

RequestIPAddr

Includes the trace file displays the IP of the server where Agent is present

IPPort

Includes the client IP port

CertSerial

Includes the certificate serial number

SubjectDN

Includes the subject DN of the certificate

IssuerDN

Includes the Issuer DN of the certificate

SessionSpec

Includes the SiteMinder session spec

SessionID

Includes the SiteMinder session ID

UserDN

Includes the User DN

Resource

Includes the requested resource

Action

Includes the requested action

RealmOID

Includes the realm OID

ResponseTime

Includes the average response time in milliseconds of the Policy Servers associated with a CA Web Agent or SDK Agent and API application

Note: To output the ResponseTime to a trace log, include the component Agent_Con_Manager along with the data field ResponseTime in the WebAgentTrace.conf file or other file specified in the Policy Server Configuration Object (ACO) and restart the Policy Server. The Agent_Con_Manager component, or Agent API Connection Manager, calculates the ResponseTime each time a response is received from a Policy Server and keeps a running average. To locate the ResponseTime in the trace log, search for [PrintStats].

Trace Message Data Field Filters

To focus on a specific problem, you can narrow the output of the trace log by specifying a filter based on the value of a data field. For example, if you are having problems with an index.html page, you can filter on resources with an html suffix by specifying Resource:==/html in the trace configuration file. Each filter should be on a separate line in the file.

Filters use the following syntax:

data_field:*filter*

The following types of filters are available:

- == (exact match)
- != (does not equal)

The filters use boolean logic as shown in the following examples:

Action:!=get (all actions except get)

Resource:==/html (all resources ending in /html)

Determine the Content of the Trace Log

The WebAgentTrace.conf file determines the content of the trace log. You can control which components and data items appear in your trace log by modifying the settings of the WebAgentTrace.conf file on your web server. The following considerations apply when editing the file:

- Entries are case-sensitive.

  When you specify a component, data field, or filter, the values must match exactly the options in the WebAgentTrace.conf file instructions.

- Uncomment the configuration settings lines or they will be ignored.
- If you modify the WebAgentTrace.conf file before installing a new Web Agent over an existing Web Agent, the file is overwritten. Therefore, you should rename or back up the file first. After the installation, you can integrate your changes into the new file.

To determine the content of the trace log

1. Open the WebAgentTrace.conf file.

   Note: We recommend duplicating the original file and making changes to the copy. This preserves the default settings.

2. Add components and subcomponents using the following steps:

   a. Find the section that matches your type of agent. For example, if you have an Apache 2.0 Web Agent installed on your server, look for a line resembling the following:

      ```
      # For Apache 2.0, Apache 2.2, IIS 6.0, IIS 7.0 and SunOne Web Agents
      ```

      The SiteMinder r12.0 SP2 IIS Web Agent is an ISAPI filter and extension, which operates on both IIS 6 and IIS 7. IIS 7 installations require additional role services on your web server before configuring the SiteMinder Web Agent.

      Note: For more information, see the *SiteMinder Web Agent Installation Guide*.

   b. Locate the following line in that section:

      ```
      #components:
      ```

   c. Uncomment the line (if it is still commented), and then add the component names you want after the colon. Separate multiple components commas as shown in the following example:

      ```
      components: AgentFramework, HTTPAgent
      ```

   d. (Optional) Follow the component name with the name of a subcomponent you want. Separate the subcomponent name with a slash as shown in the following example:

      ```
      components: AgentFramework/Administration
      ```

3. Add data fields and filters using the following steps:
   a. Locate the following line in the appropriate section:

      ```
      #data:
      ```

   b. Uncomment the line (if it is still commented), and then add the data fields you want after the colon. Separate multiple data fields with commas as shown in the following example:

      ```
      data: Date, Time, Pid, Tid, TransactionID, Function, Message, IPAddr
      ```

   c. (Optional) Add filters to your data fields by following the data field with a colon, the boolean operator and the value you want. The values you specify for the filters must match exactly. The following example shows a filter which logs activities for a specific IP address:

      ```
      data: Date, Time, Pid, Tid, TransactionID, Function, Message, IPAddr:==127.0.0.1
      ```

      Note: Each filter must be on a separate line in the file.
4. Save your changes and close the file.
5. Restart the web server to apply your changes.

   The content of the trace log has been determined.

Limit the Number of Trace Log Files Saved

You can limit the number of trace logs that a Web Agent keeps. For example, if you want to save disk space on the system that stores your Web Agent Logs, you can limit the number of trace logs using the following parameter:

TraceFilesToKeep

Specifies the number of Web Agent trace log files that are kept. New trace logs are created in the following situations:

- When the Web Agent starts.
- When the size limit of the trace log (specified by the value of the TraceFileSize parameter) is reached.

Changing the value of this parameter does *not* automatically delete any existing trace logs which exceed the number that you want to keep. For example, If your system has 500 trace logs stored, and you decide to keep only 50 of those files, the Web Agent does *not* delete the other 450 trace logs.

Setting the value of this parameter to zero retains all the trace logs.

Default: 0

To limit the number of trace logs saved

1. Archive or delete any existing trace logs from your system.
2. Set the value of the TraceAppend parameter to no.
3. Change the value of the TraceFilesToKeep parameter to the number of trace logs that you want to keep.

Collect Detailed Agent Connection Data with an Agent Connection Manager Trace Log

To collect detailed information about the connections between a Web Agent and Policy Server, you create a Trace Log file that contains information gathered by the Agent Collection Manager.

To collect detailed web agent connection data

1. Open your Agent Configuration object or local configuration file.
2. Set the value of the TraceFile parameter to yes.

   Note: Setting the value of this parameter to yes in a local configuration file of a web server overrides any of the logging settings defined on the Policy Server. For example, when the value of this parameter is set to yes in a LocalConfig.conf file log files are generated even if the value of the AllowLocalConfig parameter in the corresponding Agent Configuration object on the Policy Server is set to no. Additionally, set the related trace logging parameters (that define the file name, size, and so on) in the LocalConfig.conf file to override any Policy Server trace log settings.
3. Specify the full path to the trace log file for your Agent Connection Data in the TraceFileName parameter. This is the file that contains the trace log output.
4. Set the value of the TraceConfigFile parameter to the full path of the following file:

   web_agent_home/config/AgentConMgr.conf

   web_agent_home

   Indicates the directory where the Web Agent is installed.

   Default (Windows installations): C:\Program Files\CA\webagent

   Default (UNIX installations): /opt/ca/webagent
5. Define the format the trace log file for your Agent Connection Data by setting the following parameters:

   TraceAppend

   Adds new logging information to the end of an existing log file instead of rewriting the entire file each time logging is invoked.

   Default: No

   TraceDelimiter

   Specifies the custom character that separates the fields in the trace.conf file.

   Default: No default

   Example: |

   TraceFileSize

Specifies (in megabytes) the maximum size of a trace file. The Web Agent creates a new file when this limit is reached.

Note: This feature is not supported for Apache 1.x and Sun Java System on UNIX systems. Use the default or leave this setting blank.

Default: 0 (a new log file is not created)

Example: 20 (MB)

TraceFormat

Specifies how the trace.conf file displays the messages. Choose *one* of the following options:

- default—uses square brackets [] to enclose the fields.
- fixed—uses fields with a fixed width.
- delim—uses a character of your choice to delimit the fields.
- xml—uses XML-like tags. A DTD or style sheet is *not* provided with the Web Agent.

Default: default (square brackets)

LogLocalTime

Specifies whether the logs use Greenwich Mean Time (GMT) or local time. To use GMT, change this setting to no. If this parameter does not exist, the default setting is used.

Default: Yes

6. Restart your web server so the new settings take effect.

   Detailed information about the Web Agent connections will be collected.

   Note: For SiteMinder r12.0 SP2, the BusyHandleCount and FreeHandleCount attributes are not used.

Chapter 13: Adjust Performance

This section contains the following topics:

Web Agent Cache

The Web Agent stores user session and resource information in cache memory. This technique improves the Web Agent efficiency because the Web Agent does not have to retrieve information from the Policy Server each time a user requests access.

By configuring the cache settings, you can manage how this information is stored. The size of the cache is measured by the number of cache entries. The total number of entries in each cache cannot exceed the maximum cache size specified.

Note: You have to restart the Web Server for changes in the Web Agent cache settings to take effect.

The following guidelines apply to cache management:

- When a cache is full, new entries replace the least recently used entries.
- For the resource cache, entries are removed when the value of the ResourceCacheTimeout parameter is reached.
- For the user session cache, entries are removed based on the session timeout values that you set for each realm.

SiteMinder empties cached resource information when you modify a policy. You can also empty the user and resource caches manually from the Administrative UI.

Note: For more information, see the Policy Server documentation.

Control How Long Resource Enteries Remain Cached

You can change the amount of time that resource entries will remain in the cache with the following parameter:

ResourceCacheTimeout

Specifies the number of seconds that resource entries remain in the cache. If a user tries to access a protected resource after the time interval has been exceeded, the Web Agent removes the cached entries and contacts the Policy server.

Default: 600 (10 minutes)

Note: If you change the value of this parameter, you must restart the web server to apply the change.

To change how long the resource entries remain cached, set the ResourceCacheTimeout parameter to the number of seconds you want.

Set the Maximum Resource Cache Size

You can set a maximum on the number of resource cache entries, such as Web pages, that the Web Agent tracks with the following parameter:

MaxResourceCacheSize

Specifies the maximum number of entries that the Web Agent keeps in its resource cache. An entry contains the following information:

- A Policy Server response about whether a resource is protected
- Any additional attributes returned with the response

When the maximum is reached, new resource records replace the oldest resource records.

If you set this value to a high number, be sure that sufficient system memory is available.

If you are viewing Web Agent statistics using the OneView Monitor, you may notice that the value shown for the ResourceCacheCount is greater than the value you specified for the MaxResourceCacheSize parameter. This is not an error. The Web Agent uses the MaxResourceCacheSize parameter as a guideline and the values may at times differ because the MaxResourceCacheSize parameter represents the maximum number of average-sized entries in the resource cache. The actual cache entries are most likely larger or smaller than the pre-determined average size; therefore, the effective maximum number of entries may be more or less than the value specified.

Note: For Web Agents that use shared memory, such as the Agent on Apache 1.x and the framework Agents, the cache is pre-allocated to a constant size based on the MaxResourceCacheSize value and will not grow.

Default: (Domino web servers) 1000

Default: (IIS and Sun Java System web servers) 700

Default: (Apache web servers) 750

To set the maximum resource cache size

1. Set the value of the MaxResourceCacheSize parameter to the maximum number of resources you want.
2. For framework agents, you must restart the web server to apply the change.

   The maximum resource cache size is changed.

Disable the Resource Cache

If you are protecting an application that uses dynamic, unique URLs, you may want to disable the resource cache. Since the URLs used by the application are unique, then they will not be read from the cache.

To disable the resource cache, change the value of the MaxResourceCacheSize to zero.

Set the Maximum User Session Cache Size

You can set a maximum for the number of users the Agent maintains in the session cache with the following parameter:

MaxSessionCacheSize

Specifies the maximum number of users the Agent maintains in its session cache. The session cache stores the session IDs of users who authenticate successfully. If those users access another resource in the same realm during the same session, the Agent uses the information from the session cache instead of calling the Policy Server. When the maximum number is reached, the Agent replaces the oldest user records with new user records.

Base the value of this parameter on the number of users that you expect to access and use resources for a sustained period of time. If you set this value to a high number, ensure that sufficient system memory is available.

Default: (Domino web servers) 1000

Default: (IIS and Sun Java System web servers) 700

Default: (Apache web servers) 750

To set a maximum size of the user session cache

1. Set the value of the MaxSessionCacheSize parameter to the maximum number of users you want.
2. For framework agents, you must restart the web server to apply the change.

   The maximum user session cache size is changed.

Cache Anonymous Users

You can configure the Web Agent to store anonymous user information in a cache with the following parameter:

CacheAnonymous

Specifies if the Web Agent caches anonymous user information. You may want to set this parameter in any of the following situations:

- If your web site gets mostly anonymous users and you want to store their session information.
- If your web site gets a mix of registered and anonymous users.

  You may want to disable this parameter to keep the anonymous user information from filling the cache and leaving no room for registered users.

Default: No

To store anonymous user information in cache, set the value of the CacheAnonymous parameter to yes.

Chapter 14: Use Web Agents with Proxy Servers

This section contains the following topics:

Configure Agents that Sit behind Proxy Servers

If a Web Agent will be installed behind a proxy server, you can configure the Web Agent to work with proxy servers using the following parameters:

ProxyTrust

Instructs the Web Agent for the destination server to trust the authorizations made by the proxy server. This is more efficient because the Web Agent for the destination server does not need to reauthorize users.

Default: No

ExpireForProxy

Prevents a forward proxy server from caching content (pages and potentially headers or cookies). When this parameter is set to yes (enabled), the Web Agent inserts an Expires or Cache-control header into the HTTP response. If content is not cached, subsequent requests continue to be forwarded.

When the ExpireForProxy parameter is set to yes, the Web Agent inserts the strings specified in the appropriate ProxyHeaders*suffix* parameter into the HTTP response based upon what type of request was performed.

The Web Agent adds strings into the HTTP responses as follows:

- For HTTP/1.1 requests—if the resource is auto-authorized, then the Web Agent inserts the value of all ProxyHeadersAutoAuth parameters into the HTTP response. If the resource is protected, then the Web Agent inserts the value of all ProxyHeadersProtected parameters into the HTTP response. If the resource is not protected, then the Web Agent inserts the value of all ProxyHeadersUnprotected parameters into the HTTP response.
- For HTTP/1.0 requests—if the resource is auto-authorized, then the Web Agent inserts the value of all ProxyHeadersAutoAuth10 parameters into the HTTP response. If the resource is protected, then the Web Agent inserts the value of all ProxyHeadersProtected10 parameters into the HTTP Response. If the resource is not protected, then the Web Agent inserts the value of all ProxyHeadersUnprotected10 parameters into the HTTP response.

Default: No

Note: This parameter applies to proxy servers only.

To tell the proxy not to cache the pages, the Web Agent adds an Expires header for the page. This header is set to a date in the past, which prevents the page from being cached by a proxy, as dictated by the HTTP 1.0 specification. On 302 redirects, a cache-control: no-cache header is set instead. Although this prevents caching of content, this has the negative consequence of affecting the browsing experience for an Internet Explorer (IE) browser, as described by Microsoft Support.

With the use of cache-control: no-cache for 302 redirects, the ActiveX component that manages in-place document viewing in IE relies on the browser's cache to locate the file. Because this header instructs the browser not to cache the file, the ActiveX component cannot locate the file and fails to display the request properly. Further, when you set the Web Agent's ExpireForProxy setting to yes, the back-end server tells the proxy not to cache the resource.

To configure Agents that sit behind proxy servers

1. Set the ProxyTust parameter to yes.
2. Set the ExpireForProxy parameter to yes.
3. (Optional) Customize the cache-control and ExpireForProxy (HTTP) headers.

   The Agents behind the proxy servers are configured.

Customize the Cache-Control and ExpireForProxy Header Settings

You can customize the cache-control and ExpireForProxy headers to secure Web resources without affecting in-place activation of application files (.doc, .pdf, and so on). You can set specific HTTP headers for the following types of content independently to further characterize the data handled by the proxy server:

- Auto-Authorized
- Unprotected
- Protected

Important! We recommend using the default settings unless you are familiar with the ramifications of changing these settings in accordance with RFC 2068. If you plan to change the default settings, note that the SiteMinder session cookie is updated on access of an unprotected page once a user has a session in order to track idle timeout. Therefore, unprotected pages should not be cached on a proxy that caches HTTP headers.

The following characteristics apply to setting headers to prevent caching by proxies:

- All redirects set a Cache-Control: no-cache header, regardless of agent activity.
- The web server sends the appropriate headers back to the proxy/client based on the HTTP protocol used (1.0 or 1.1 and higher).

All parameters should be configured using multi-value strings to suit the use of multiple headers, such as cache-control: private and cache-control: max-age=60.

The following is the new configuration:

1. ProxyHeadersDefaultTime - defaults to 60 seconds
2. ProxyHeadersTimeoutPercentage – defaults to 10 percent
3. Auto-authorized resources:
   - For HTTP/1.1, configure ProxyHeadersAutoAuth parameter(s):

   ProxyHeadersAutoAuth

   Note: You must add this parameter manually.

   Default: Expires: Thu, 01 Dec 1994 16:00:00 GMT

   Example (suggested setting):
   ProxyHeadersAutoAuth="Cache-control: max-age=60"

- For HTTP/1.0, configure ProxyHeadersAutoAuth10 parameter(s):

ProxyHeadersAutoAuth10

Note: You must add this parameter manually.

Default: Expires: Thu, 01 Dec 1994 16:00:00 GMT

Example (suggested setting): ProxyHeadersAutoAuth10="Expires: Thu, 01 Dec 1994 16:00:00 GMT"

4. Unprotected content:

- For HTTP/1.1, configure ProxyHeadersUnprotected parameter(s):

ProxyHeadersUnprotected

Note: You must add this parameter manually.

Default: Expires: Thu, 01 Dec 1994 16:00:00 GMT

Cache-Control: no-cache

Example (suggested setting):
ProxyHeadersUnprotected="Cache-Control: private"

ProxyHeadersUnprotected="Cache-Control: max-age=60"

- For HTTP/1.0, configure ProxyHeadersUnprotected10 parameter(s):

ProxyHeadersUnprotected10

Note: You must add this parameter manually.

Default: Expires: Thu, 01 Dec 1994 16:00:00 GMT

Cache-Control: no-cache

Example (suggested setting): ProxyHeadersUnprotected10="Expires: Thu, 01 Dec 1994 16:00:00 GMT"

5. Protected content:

- For HTTP/1.1, configure ProxyHeadersProtected parameter(s):

ProxyHeadersProtected

Note: You must add this parameter manually.

Default: Expires: Thu, 01 Dec 1994 16:00:00 GMT

Cache-Control: no-cache

Example (suggested settings):
ProxyHeadersProtected="Cache-Control: private"

ProxyHeadersProtected="Cache-Control: max-age=60"

- For HTTP/1.0, configure ProxyHeadersProtected10 parameter(s):

ProxyHeadersProtected10

Note: You must add this parameter manually.

Default: Expires: Thu, 01 Dec 1994 16:00:00 GMT

Cache-Control: no-cache

Example (suggested settings): ProxyHeadersProtected10="Expires: Thu, 01 Dec 1994 16:00:00 GMT"

When configuring multiple headers, (for example, the cache-control headers in the suggested setting for unprotected HTTP/1.1 content), note the following:

- You *must* have multiple occurrences of the configuration parameter and you cannot separate these with a comma (,) or the plus-sign (+).
- As the values for these configuration parameters are HTTP response headers, they must comply with RFC 2616 (for HTTP/1.1), RFC 1945 (for HTTP/1.0) and RFC 822. Both HTTP/1.1 and HTTP/1.0 specify the format for an HTTP Header as that of an RFC 822 message, namely "Name: Value" (Name, followed by a colon, white space and then a value).

If you do not configure the Web Agent to set the appropriate cache expiration headers when a user accesses unprotected resources, then by default, the Web Agent will not set these headers, thereby allowing a proxy (or browser) to cache an SMSESSION cookie. This cached cookie can be re-used by the proxy (or browser) after the user has initiated a different session (and therefore a different user context), causing an unauthorized impersonation.

Proxy Header Usage Notes

- To prevent the Web Agent from sending any proxy headers, blank out the ProxyHeadersUnprotected value. For example:

  ProxyHeadersUnprotected=""

  Note: To get a double quote character (") to appear, use a single quote (`). The Web Agent automatically converts it to a double quote.
- The value, %% or %d (treated identically) may appear within a ProxyHeaders line. This value is replaced with either the smaller of the IdleTimeout and SessionTimeout multiplied by the ProxyHeadersTimeoutPercentage, or, if the timeouts are not set, the ProxyHeadersDefaultTime is used.
- Ensure that values for the standard (1.1 and higher) and HTTP 1.0 headers are set properly for requests to the back-end server.
- ExpireForProxy="YES" will expire cookie provider redirects carrying the SMSESSION cookie in the query string.

Security Considerations

Browser sessions can persist after logout, so removing the SMSESSION cookie does not prevent a user from using the same browser session to view previously cached files. This problem occurs because the proxy server is not aware of the logout request and retains any protected/unprotected content in cache for the cache-control: private user until it timed out (cache-control: max-age=60). Thus, such a request would result in a page returned with a valid SMSESSION cookie. The only way to ensure security is to disable keep-alives or close the browser.

Further, the local browser cache is affected by the private/max-age combination since it observes local cache across sessions. For this reason, the max-age time for protected resources should be as short as possible.

Employing the if-modified-since and if-none-match request headers when the allowcacheheaders="FALSE" configuration setting is used (default) does not prevent the proxy server from observing these headers. Thus, these observed headers take effect on the request according to the proxy server.

You could work around this issue by installing:

- a Web Agent on the proxy server.
- another filter that removes these headers from the request.

Since HTTP 1.0, HTTP 1.1, or higher use different headers for specifying instructions to caching proxies, these versions should be configured in a way to ensure the most appropriate handling based on the type of connection.

Chapter 15: Configure Reverse Proxy Servers

This section contains the following topics:

Types of Reverse Proxy Solutions

SiteMinder supports the following reverse proxy solutions:

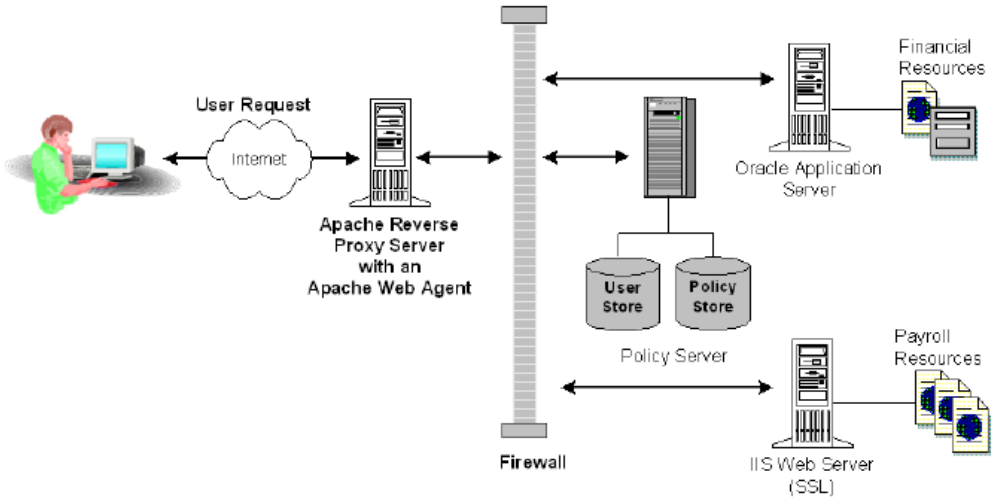
- The Apache-based reverse proxy agent
- The Sun Java System-based reverse proxy agent
- The SiteMinder Secure Proxy Server

How Reverse Proxy Servers Work with SiteMinder

You can configure the Apache or Sun Java System web server to function as a *reverse proxy server*. A reverse proxy server is a proxy server that acts on behalf of an enterprise to forward requests to an organization's internal network. It allows clients to access resources on backend servers, which are servers behind a firewall.

If your environment uses an Apache or Sun Java System reverse proxy server as a gateway to your back-end servers, a SiteMinder Web Agent can protect these resources. Therefore, you can protect resources that are not protected by a backend SiteMinder Web Agent. Also, your resources are secure for intranet and authorized Internet users.

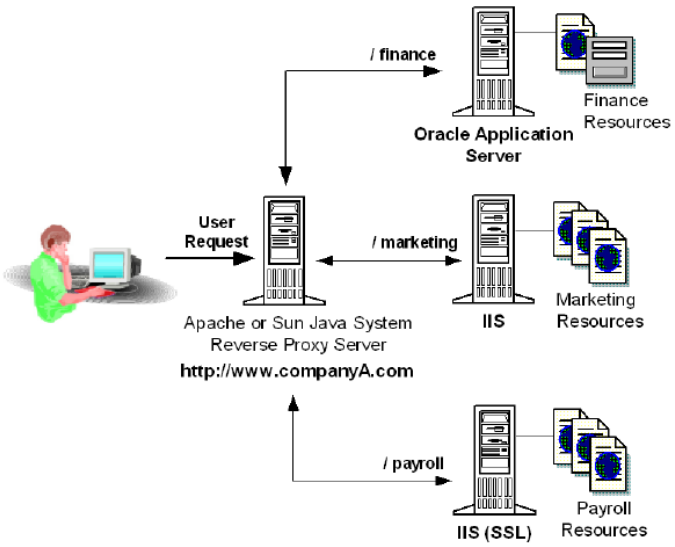
The following illustration shows a network with a reverse proxy server:



Reverse proxy servers provide the following advantages:

- Users within a cookie domain may seamlessly access resources on backend servers. Users from other domains must authenticate through the reverse proxy server and typically, a firewall before gaining access to those same backend servers.
- Users can access different resources hosted on several backend servers using the same domain name.
- Reverse proxy agents support the same features as other SiteMinder Web Agents for Apache or Sun Java System Web servers.
- If you have resources located on servers for which a SiteMinder Agent is not yet available, you can implement an Apache or Sun Java System reverse proxy server in front of the servers. The Web Agent for the Apache or Sun Java System server then protects everything on the backend.

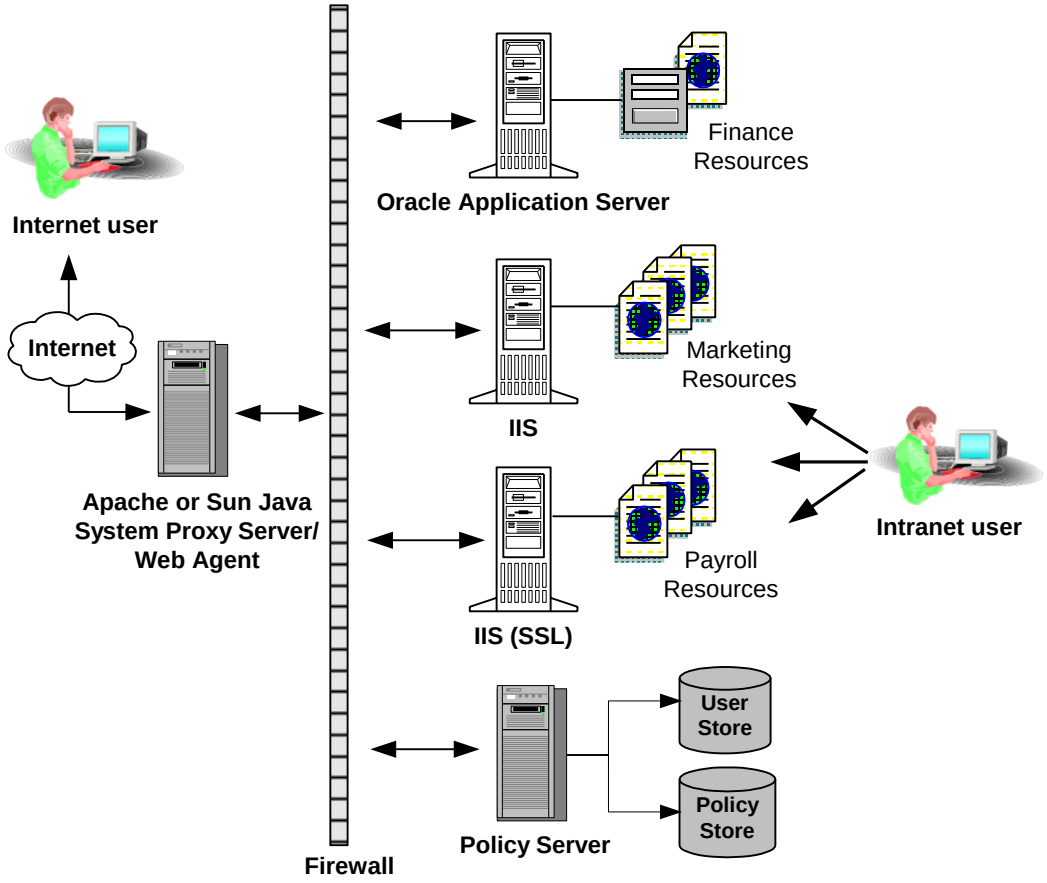
The following illustration describes how a reverse proxy server forwards requests from users to the appropriate resource on a back end server:



SiteMinder Reverse Proxy Deployment Considerations

Typically, when you deploy an Apache or Sun Java System reverse proxy Agent, a firewall is located between the Apache or Sun Java System Web Agent and the servers hosting the protected resources. The Policy Server should also be located behind the firewall.

The following illustration shows a SiteMinder reverse proxy deployment.



When deploying a SiteMinder reverse proxy Agent, consider the following:

- If a policy has been configured to return response attributes, the variables are sent to both the reverse proxy server and the backend web server on which the protected resource resides. When a request is made for a protected resource, the Policy Server first sends response attributes (CGI or HTTP variables) to the Agent on the Apache or Sun Java System server. The Agent then puts the response attributes in the request that is sent to the backend server.
- If any of the backend servers or protected applications provide their own authentication functionality, the authentication must be disabled. Disabling the backend authentication ensures that SiteMinder's authentication takes precedence.

  Important! When configuring the cache for the reverse proxy be aware that all cookies are cached, including the SMSESSION cookie. For assistance contact your Apache or Sun Java System vendor for support.

More Information

Define HTTPS Ports (see page 137)

How to Configure An Apache Reverse Proxy Server

To configure an Apache-based reverse proxy server with SiteMinder, use the following process:

1. Update the settings of the following parameters for any Apache Web Agents located *behind* the Apache reverse proxy server:
   a. Set the ProxyAgent value to yes to indicate that this Agent is acting as a reverse proxy Agent.
   b. Set ProxyTimeout parameter to a value (in seconds).

      The reverse proxy uses this value to time out the requests it makes to the Web Agent deployed behind it.
   c. Edit the BadURLChars parameter by removing all occurrences of the following value from the list:

      %
   d. (Optional) Enable the ProxyTrust parameter.

      By setting this parameter to yes, it instructs the Web Agent behind the proxy Agent to trust the session information sent from the proxy Agent. and not to re-validate it. Enabling this parameter makes communication more efficient because only one call is made from the proxy Agent to the Policy Server. The Agent behind the proxy does not have to contact the Policy Server.
   e. Set the httpsports parameter to indicate to the Apache server which port is set up for SSL.
2. Add the following directives to the httpd.conf file of your Apache Web server:

   ProxyPass

   Allows mapping of remote servers to the local server. The values in this directive use the following format:

   /local_virtual_path partial_URL_of_remote_server

   Example: ProxyPass /realma/ http://server.example.org/realma/

   ProxyPassReverse

   Allows adjustment of the location header by the Apache server on HTTP redirect responses. The values in this directive use the following format:

   /local_virtual_path partial_URL_of_remote_server

   Example: ProxyPassReverse /realma/ http://server.example.org/realma/

   Note: For more information about directives, see the documentation for your web server.
3. Restart your Apache web server.

Configure a Sun Java System 6.0 Reverse Proxy Server

You can use a Sun Java System 6.0 Web server as a reverse proxy with SiteMinder.

To configure a Sun Java System Web server as a reverse proxy

1. Navigate to the following directory:

   web_server_installation_directory/plugins

2. Create the following subdirectories:

   \passthrough\bin

3. Add the one of the folloiwng files to the passthrough\bin directory:

   - passthrough.dll (Windows)
   - libpassthrough.so (UNIX)

   Note: Download the Sun Java System Web Server Reverse Proxy Add-On 6.1 file from Sun Microsystems.

4. Add the one of the following sections to your *web_server_installation_directory*/config/magnus.conf file:

   Windows

   Init fn="load-modules" shlib="*web_server_installation_directory*/plugins/passthrough/bin/passthrough.dll" funcs="init-passthrough,auth-passthrough,check-passthrough,service-passthrough" NativeThread="no"

   Init fn="init-passthrough"

   UNIX

   Init fn="load-modules" shlib="*web_server_installation_directory*/plugins/passthrough/bin/libpassthrough.so" funcs="init-passthrough,auth-passthrough,check-passthrough,service-passthrough" NativeThread="no"

   Init fn="init-passthrough"

   Note: All lines *must* start with Init. Any settings that follow must continue on the *same* line.

5. Create a passthrough object by adding the following section to top of the config/obj.conf file:

   <Object name="passthrough">
   ObjectType fn="force-type" type="magnus-internal/passthrough"
   Service type="magnus-internal/passthrough" fn="service-passthrough" servers="http://*server_name*:*port*"
   Error reason="Bad Gateway" fn="send-error" uri="$docroot/badgateway.html"
   </Object>

6. Configure which URIs are to be forwarded by adding a section similar to the following to the top of the default object in the server_install_directory/config/obj.conf file:

   ```
   NameTrans fn="assign-name" from="(/uri|/uri/*)" name="passthrough"
   ```

   The URI is the context root of a web application deployed on the remote servers, and passthrough corresponds to the name of the <Object> in the obj.conf file, as shown in the following example:

   ```
   <Object name="default">
   ...
   NameTrans fn="assign-name" from="(/webapp1|/webapp1/*)" name="passthrough"
   ...
   </Object>
   ```

   The name value should match the value of the object name used in Step 5.

7. Restart the Web server.

   The reverse proxy is configured.

Configure a Sun Java System 7.0 Reverse Proxy Server

You can use a Sun Java System 7.0 Web server as a reverse proxy with SiteMinder.

To configure a Sun Java System Web server as a reverse proxy

1. Add the following directive to the obj.conf file:

   NameTrans

   Specifies the local and remote virtual paths using the following format:

   NameTrans fn="map" from="*local_virtual_path*" name="reverse-proxy-/local_virtual_path" to="*remote_virtual_path*"

   Example: NameTrans fn="map" from="/realma" name="reverse-proxy-/reamla" to="http:/realma"

2. Add the following directives at the *end* of the obj.conf file:

   Object name

   Specifies the name of the local virutal path and the URL of the remote virutal path used in the NameTrans directive, using the following format:

   <Object name="reverse-proxy-/*local_virtual_path*">

   Route fn="set-origin-server" server="http://*remote_server_URL*:*port*"

   </Object>

   Example: <Object name="reverse-proxy-/reamla">

   Route fn="set-origin-server" server="http://server.example.org:*port*"

   </Object>

   Object ppath

   Specifies the partial path given to the server by the client.

   Example: <Object ppath="http:*">

   Service fn="proxy-retrieve" method="*"

   </Object>

3. Restart the web server.

   The reverse proxy is configured.

SiteMinder Secure Proxy Server

For users who require a more sophisticated reverse proxy solution, the SiteMinder Secure Proxy Server provides the following benefits over the Apache or Sun Java System-based SiteMinder Reverse Proxy Agent:

- An embedded and fully supported web server, including SSL accelerator card support and a GUI tool for managing keys and certificates
- Support for multiple session schemes (cookie-based, and cookie-less)
- Support for flexible proxy rules, such as the following:
  - Support for rules based on HTTP headers and SiteMinder responses, in addition to URLs
  - Ease of use for complex rules

The Secure Proxy Server introduces a new layer in the traditional SiteMinder architecture. This layer forwards or redirects all requests to destination servers in the enterprise.

When the Secure Proxy Server processes a request, the URL requested by the user is preserved in an HTTP header variable called SM_PROXYREQUEST. This header may be used by other applications that require the original URL requested by a user before the Secure Proxy Server proxied the request.

SM_PROXYREQUEST HTTP Header for SiteMinder Processing with Secure Proxy Server

The Secure Proxy Server introduces a new layer in the traditional SiteMinder architecture. This layer forwards or redirects all requests to destination servers in the enterprise.

When the Secure Proxy Server processes a request, the URL requested by the user is preserved in an HTTP header variable called SM_PROXYREQUEST. This header may be used by other applications that require the original URL requested by a user before the Secure Proxy Server proxied the request.

Chapter 16: Control Inbound URL Processing

This section contains the following topics:

Decode Query Data in a URL

To have the Web Agent's Base64 algorithm decode a URL's query data before calling the Policy Server (so the Policy Server sees the proper resource), use the following parameter:

DecodeQueryData

Specifies whether the Web Agent decodes the query data in a URL before calling the Policy Server. Set this parameter to yes if you need do any of the following tasks in your environment:

- If you need to ensure the rules filer acts against the proper string.
- If you need to or write rules against the data in a query string.

Default: No

To have the Web Agent decode the query data in a URL before calling the Policy Server, set the value of the DecodeQueryData parameter to yes.

Ignore Query Data in a URL

The IgnoreQueryData parameter affects the way Web Agents treat URLs. If you do not want the Web Agent cache the entire URL and send the URIs with their query strings to the Policy Server for rule processing, you improve performance with the following parameter:

IgnoreQueryData

Specifies whether the Web Agent will cache the entire URL (including the query strings) and send the entire URI to the Policy Server for rule processing. A full URL string contains a URI, a hook (?), and some query data, as shown in the following example:

URI?*query_data*

URLs that have been the subjects of requests are cached by default. Subsequent requests search the cache for a match. If requests for the same URI contain different query data, the match fails. Ignoring the query data improves performance.

When the IgnoreQueryData parameter is set to yes, the following occurs:

- The URL is truncated at the hook. Only the URI is cached and sent to the Policy Server. The query data is maintained elsewhere, for the purpose of maintaining the proper state for redirects.
- Only the part before the hook is sent to the Policy Server for rule processing.
- Both URIs in the following example are handled as the same resource:

  /myapp?data=1

  /myapp?data=2

When the IgnoreQueryData parameter is set to no, the following occurs:

- The entire URL is cached.
- The entire URI is sent to the Policy Server for rule processing.
- The URIs in the following example are handled as different resources:

  /myapp?data=1

  /myapp?data=2

Default: No

To have the Web Agent send only URIs to the Policy Server for processing, set the value of the IgnoreQueryData parameter to yes.

Important! Do not enable this setting if you have policies which depend on URL query data.

Query String Encryption of Redirect URLs

When a Web Agent communicates with credential collectors, such as the FCC and SCC, the Password Services application (CGI or JSP), or a Cookie Provider, it uses protocol parameters that are shown in clear text in the redirection URL.

The Web Agent can now encrypt all SiteMinder query parameters in a redirect URL, further securing Agent interactions.The Web Agent is only encrypts data sent between SiteMinder components, not for redirects to non-SiteMinder applications.

When query string encryption is enabled, the Web Agent encrypts query data when it returns a 302 redirect response to the browser. The 302 response redirects the user to another SiteMinder resource.

All the query parameters are grouped into a single query parameter called smquerydata. When the SecureUrls parameter is enabled, SiteMinder denies access to any request that does not have an encrypted smquerydata parameter, where required.

The SecureUrls feature is not supported when any of the following parameters are enabled:

FCCCompatMode

Enable an FCC/NTC to serve up forms for resources protected by 4.x Web Agents or third party applications.

Note: SMUSRMSG is supported for the custom authentication scheme only when FCCCompatMode set to yes.
Default: (traditional agents) Yes

Default: (framework agents) No

Important! Setting this parameter to no removes support for version 4.x of the Netscape browser.

LegacyEncoding

Forces the Web Agent to replace any dollar sign ($) characters in legacy URLs with a hyphen (-). This also ensures backwards comparability with MSR, Password Services, and DMS. When this parameter is set to no, a Web Agent converts the string SM to -SM-. When this parameter is set to yes, the Web Agent does *not* convert the dollar sign ($) character.

Default: (Framework Agents) No

Default: (Traditional Agents) Yes

If the SecureUrls parameter is set to yes, the Web Agent ignores the values of the previous parameters, even if they are set to yes. When this happens, these parameters have a value of no in the Agent logs, regardless of their settings in the configuration object or configuration file, as shown in the following example:

```
[12/Jul/2005:05:23:57-975-1-0] SecureUrls: 'YES'
[12/Jul/2005:05:23:57-975-1-0] FccCompatMode: 'NO'
[12/Jul/2005:05:23:57-975-1-0] LegacyEncoding: 'NO'
```

Query String Encryption of Redirect URLs and Credential Collectors

When you encrypt the query strings of redirect URLs use credential collectors, the credential collectors provide the keys used to encrypt the query data.

For forms authentication schemes, the query string directive, smquerydata, is part of the FCC template. The Web Agent serving the FCC uses this directive to send the encrypted query data to the target Web Agent when the FCC is posted.

The following directive is used:

```
<INPUT type='hidden' name='smquerydata' value='$$smquerydata$$'>
```

Note: If you are using custom FCCs, you must add the smquerydata directive along with other FCC directives, such as TARGET to the custom FCC.

SiteMinder r12.0 SP2 Web Agents with the SecureUrls parameter enabled can operate only with credential collectors served from other Web Agents that support this functionality, which was introduced at 5.x QMR 7.

Query String Encryption of Redirect URLs and FCC-based Password Services

If you want to encrypt the query strings of redirect URLs, you can only use the FCC-based Password Services; CGI or JSP-based Password Services will not work with encrypted query parameters. If you set the SecureUrls parameter to no, you can use any one of the three Password Services versions.

Note: CGI and JSP Password Services are deprecated as of 5.x QMR 7, but are still supported.

Encrypt Query String Parameters in Redirection URLs

The following parameter enables the Web Agent to encrypt all SiteMinder query parameters in a redirect URL:

SecureURLs

Specifies whether the Web Agent encrypts the SiteMinder query parameters in a redirect URL. You can use this setting to provide additional security for requested resources protected by an advanced authentication scheme, Password Services, or when a request invokes the Cookie Provider.

Important! The Web Agent only encrypts data sent between SiteMinder components. The data sent for redirects to non-SiteMinder applications is not encrypted.

The following SiteMinder credential collectors and applications support the SecureUrls functionality:

- HTML Forms authentication
- Cert And Forms authentication
- SSL Authentication
- Cert or Forms authentication
- NTLM authentication
- ACE authentication
- SafeWord authentication
- User self registration
- Multi-domain Single Sign-on with Cookie Provider
- FCC-based Password Services (not CGI- or JSP-based)

Default: No

To encrypt query string parameters in redirection URLs

1. Set the value of the SecureURLs parameter to yes.
2. If you want to encrypt query string parameters in redirection URLs within a single sign-on environment, ensure all of the Web Agents in the single sign-on environment have the SecureURL parameter set to the same value.
3. If you are using custom FCCs, add the smquerydata directive along with the other FCC directives (such as TARGET) to the custom FCC.

   Query string parameters are encrypted in SiteMinder redirection URLs.

Allow Un-restricted Access to URIs

If you have URIs that you do not want to protect with SiteMinder, you can direct the Web Agent to ignore and allow un-restricted access to those URIs by setting the following parameter:

IgnoreUrl

Specifies a URI within a URL that will not be protected. Users attempting to access the resource associated with the URI will not be challenged. The Web Agent ignores the URI portion of the string after three forward slashes. For example, if you set this parameter to the following value:

http://www.example.com/directory

The Web Agent ignores the following URI:
directory

The Web Agent ignores the specified URI wherever it occurs, even if it is under a different domain. For example, the Web Agent ignores the URI shown previously in all of the following URLs:

http://www.example.com/directory
http://www.example.net/directory
http://www.example.org/directory

Note: This value is case-sensitive.

Default: No default.

Example: (multiple URIs in local configuration file)

IgnoreUrl="http://www.example.com/directory"

IgnoreUrl="http://www.example.com/directory2"

Example: (using a URI only, without specifying a domain)

IgnoreUrl="/*resource*/"

To allow un-restricted access to URIs, do either of the following tasks:

- For central configuration, add the fully qualified domain names with the URIs that you want to ignore to your agent configuration object. For more than one URI, use the multi-value setting for the parameter.
- For local configuration, add a separate line for each fully qualified domain name and URI in the local configuration file.

Resources using the specified URIs are ignored by the Web Agent and access to those resources is granted automatically.

Set a Maximum URL Size

You can increase the maximum URL size that a Web Agent can handle with the following parameter:

MaxUrlSize

Specifies the maximum size (in bytes) of a URL that a Web Agent can handle. Because different web servers have different limitations on URL length, check the documentation from your web server vendor before setting this parameter.

Default: 4096 B

To change the maximum URL size, change the number of bytes specified in the MaxUrlSize parameter.

Chapter 17: Enforce Security with URL Monitoring

This section contains the following topics:

URL Monitoring Overview

The Web Agent can prevent attacks by malicious users trying to halt normal operation of a Web site or circumvent a site's security mechanisms to gain illegal access to information.

The Web Agent monitors URLs in resource requests and enforces the security policies for these resources. SiteMinder Web Agents interpret and parse URLs differently from the web servers where the resources reside. These differences can result in subtle performance and security issues that potentially allow unauthorized users to gain access to resources. You need to consider these issues in the design of your Web site and the configuration of the SiteMinder Web Agent.

Reduce Overhead by Ignoring File Extensions of Unprotected Resources

You can reduce SiteMinder overhead by instructing the Web Agent to ignore requests for certain types of resources with the following parameter:

IgnoreExt

Specifies the types of resources for which the Web Agent passes requests to the web server without checking SiteMinder policies.The Web Agent allows access to the items specified by this parameter even if they exist in a realm that is protected by a SiteMinder policy.

Requests for resources that meet either of the following conditions may be ignored:

- The resource ends in one of the extensions that you configure the Web Agent to ignore.
- The URI of the protected resource contains a single period (.).

  For example, if a URI for a requested resource is /my.dir/ the Web Agent passes the request directly to the web server.

Default: .class, .gif, .jpg, .jpeg, .png, .fcc, .scc, .sfcc, .ccc, .ntc

Important! Use caution when setting the IgnoreExt parameter. There are some security issues that you may want to consider.

By default, the Agent does *not* ignore requests for resources that contain two or more periods separated by a slash (/). Web Agents handle requests for resources using the process shown in the following example:

1. The .gif extension is added to the IgnoreExt parameter. Requests for resources with the .gif extension are be ignored by the Web Agent.
2. A request is made for the following URI:

   /dir1/app.pl/file1.gif,
3. The Web Agent checks /dir1/app.pl/file1.gif against the policy server because some web servers will execute /dir1/app.pl as an application instead of serving the file1.gif resource.

   Granting access to /dir1/app.pl/file1.gif without consulting the web server may have caused a security breach.

To reduce overhead by ignoring the file extensions of unprotected resources, add the extensions of the resources you want to ignore to the value IgnoreExt parameter.

How to Protect Resources Without Periods or Extensions

Some URLs, such as servlets, do not have peroids. Other URLs may not have extensions. Both of these situations pose security risks. The following process demonstrates these risks:

1. Your environment contains a directory called /mydir/servlets that is a protected resource.
2. Your Web Agent is configured to ignore requests for resources with the .gif extension.
3. An unauthorized user appends the name of a nonexistent file along with a .gif extension to the end of the URL as shown in the following example:

   /mydir/servlets/file.gif
4. The Web Agent ignores the .gif extension and grants the unauthorized user access to the /mydir/servelets directory.

If you are most concerned about the security risks, do not allow the Agent to ignore any extensions, but consider the following consequences:

- Performance may decrease because the Web Agent will evaluate every image URL on a page.
- Behavior of your Web site may change because users may be challenged for resources that formerly did not require authentication.

The following options are available to protect URLs that do not have periods:

- Configure the Agent to use the OverrideIgnoreExtFilter feature.
- Make sure that protected resources do not have extensions that the Web Agent is configured to ignore.

Protect Resources Without Extensions

To prevent unauthorized users from gaining access to resources without extensions, you can use the following parameter:

OverrideIgnoreExtFilter

Specifies a list of strings you want the Web Agent to match against all URIs. This helps you protect resources whose extensions are normally ignored by the Web Agent, or any files or applications that do not have extensions. If the URI matches one of the strings in the list, the Web Agent checks with the Policy Server to determine if the resource is protected.

It is better to specify more general strings instead of exact paths. You can also include a partial string to protect a group of resources. For example, the string /servlet/ protects the following resources:

- /dira/app1/servlet/app
- /dirb/servlet/app1
- /dirc/mydir/servlet/app2

Default: No default

To protect resources without extensions, add strings for the resources (without periods) that you want to protect to the value of the OverrideIgnoreExtFilter parameter. If you are using an Agent Configuration Object, use the multi-value option to add the strings. If you are using a local configuration file, add each string on its own line.

Secure Applications

An unauthorized user can append a false file name that contains an extension that the Web Agent is configured to ignore to the end of a URL . The Agent then allows the unauthorized user access to the resource. To have the Web Agent deny access to such attempts, use the following parameter:

SecureApps

Prevents the Agent from authorizing URLs from an unauthorized user. If your Web Agent is configured to ignore requests for files ending with certain extensions, an attacker may attempt to access resources by creating a false URL.

For example, if you have a resource with the following URL:

/scripts/myapp

An attacker may attempt to gain access by creating a false URL like the one in the following example:

/scripts/myapp/junk.jpg

If the value of the SecureApps parameter is set to no, the request for /scripts/myapp/junk.jpg would be automatically authorized if the Web Agent was set to ignore requests for .jpg files.

If the value of the SecureApps parameter is set to yes, the Web Agent attempts to discover if the resource is legitimate or if the URL is false.

Default: No

To secure applications, set the value of the SecureApps parameter to yes.

Handle Complex URIs

The DisableDotDotRule parameter determines whether or not the Web Agent automatically authorizes a URI that contains two dots separated by a slash (/).

If the DisableDotDotRule is set to yes, the Agent *does not* apply the double dot rule. For example, if the URI is:

- /dir1/app.pl/file1.gif

  The Web Agent uses the IgnoreExt parameter to determine if the resource should be automatically authorized.

- /dir1/okay.button.gif

  The Agent can ignore this URI because the two dots are not separated by a slash (/). The double-dot rule is not applicable in this case.

If the DisableDotDotRule is set to no, the default, the Web Agent *applies* the double-dot rule. The Web Agent challenges requests for the following URIs, passing the request to the Policy Server:

- /dir1/app.pl/file1.gif

  This URI falls under the double-dot rule because the two dots are separated by a slash.

  The web server may consider /dir1/app.pl as the target resource, and /file1.gif as extra path information, typically viewable in CGI headers as PATH_INFO.

- /dir1/okay.button.gif

  The Agent may ignore this URI because even though the double-dot rule is being enforced, the two dots are not separated by a slash (/), so the rule is not applicable.

Important! Avoid creating the possibility for unauthorized access when you use the IgnoreExt and DisableDotDotRule parameters together. For example, if you want to protect /dir1/app.pl, but you set the DisableDotDotRule parameter to yes, the Agent ignores the URI /dir1/app.pl/file1.gif because you have disabled the double-dot rule and included .gif in the IgnoreExt parameter. Consequently, an unauthorized user may access the protected application /dir1/app.pl.

Specify Bad URL Characters

You can list a set of character sequences that cannot be part of a URL request. These are treated by the Agent as bad URL characters. The Web Agent will refuse URL requests that contain any of the characters or strings of characters that you include in this list. The checking is done on the URL before the "?" character. The Web Agent rejects URL requests that include such characters because a malicious Web client might use such characters to evade SiteMinder rules.

When a Web Agent refuses a URL request containing a Bad URL character, the web server responds with one of the following messages:

- Internal Server Error
- Web Page not Found (404) Error

Check your Web Agent logs for information on how the Agent is handling requests.

You specify the characters with the following parameter:

BadUrlChars

Specifies the character sequences that cannot be used in URL requests. The Web Agent checks the characters in the URL that occur before the "?" character against those specified by this parameter. If any of the specified characters are found, the Web Agent rejects the request.

You can specify the following characters:

- a backward slash (\)
- two forward slashes (//)
- period and a forward slash (./)
- forward slash and a period (/.)
- forward slash and an asterisk (/*)
- an asterisk and a period (*.)
- a tilde (~)
- %2d
- %20
- %00-%1f
- %7f-%ff
- %25

Separate multiple characters with commas. Do *not* use spaces.

You can use the bad URL characters in CGI parameters if the question mark (?) precedes the bad URL characters.

Default: <,>,&,;

Limits:

- You can specify characters literally or enter the URL-encoded form of that character. For example, you can enter the letter a, or you can enter the encoded equivalent of %61.
- You can specify a maximum number of 4096 characters (including commas used for separating characters).
- You can specify ranges of characters separated by a hyphen. The syntax is: *starting_character-ending_character*. For example, you can enter a-z as a range of characters.
- Specify quotes (") with the URL-encoded equivalent of %22. Do *not* use ASCII.

To specify Bad URL characters, edit the value of the BadURLChars parameter to include the characters that you want to block.

Note: When configuring the Apache 2.0 Reverse Proxy Server and Outlook Web Access (OWA), be sure to turn off the BadURLChars parameter. OWA allows unrestricted characters in the email subject that might be listed in the BadURLChars parameter.

More Information

Types of Reverse Proxy Solutions (see page 195)

IIS 6.0 Servers and BadURLChars Settings

The SiteMinder Web Agent for an IIS 6.0 web server functions as an ISAPI extension. When a HTTP request comes through, IIS 6.0 web server always processes the request first before the Web Agent.

The IIS 6.0 web server may filter a URL for bad characters before passing the request to the Web Agent. The server maps a URI to a physical Web resource, such as an HTML page or CGI application. As a result, the Web Agent may not see certain characters from the original URI if the URI is modified during the mapping process. The Web Agent only acts on the resource that the web server passes to it. You need to consider this when including characters in the BadURLChars parameter.

Note: After the IIS 6.0 web server filters some characters, it may return an error page instead of passing the request to the Web Agent.

Specify Bad Form Characters

The bracket, ampersand, and quotation characters, <, >, &, " are commonly used in cross-site scripting attacks. The BadFormChars parameter specifies which of these characters the Web Agent should encode as literal HTML characters before outputting them to an HTML form. The encoding allows the intentional use of scripting code for presenting forms for an authentication challenge.

There are only four valid characters for this setting, which are all set by default: <, >, &, ". Adding characters other than these four has no effect.

Note: The quotation mark (") must be entered as %22, the hexidecimal equivalent of the ASCII character.

If your application uses custom forms and you want these characters to remain unencoded in raw HTML, leave the BadFormsChars parameter blank or set it as follows:

Parameter Setting	Result
BadFormsChars=""	No characters are encoded
BadFormsChars="<,>,%22,&"	All four characters are encoded < is encoded as &lt > is encoded as &gt & is encoded as &amp " (%22) is encoded as &quot
BadFormsChars="<,>"	Only < and > characters are encoded
BadFormsChars="%22"	Only the quotation mark (") is encoded

Note: There are no spaces between the opening and closing quotes and the value the quotes enclose.

Only directive substitutions are encoded as raw HTML—the source lines in the form template, such as login.fcc are unchanged. Keeping the source lines unchanged prevents dynamic data containing scripting code from being sent back to the browser as data in the form.

Specify Bad Query Characters

To prevent certain characters the query string portion of a URL, set the following parameter:

BadQueryChars

Specifies characters that the Web Agent prohibits in the query string portion (following the '?') in a URL.

Default: Empty (any characters allowed in query strings)

Limits:

- You can specify characters literally or enter the URL-encoded form of that character. For example, you can enter the letter a, or you can enter the encoded equivalent of %61.
- You can specify a maximum number of 4096 characters (including commas used for separating characters).
- You can specify ranges of characters separated by a hyphen. The syntax is: *starting_character-ending_character*. For example, you can enter a-z as a range of characters.
- Specify quotes (") with the URL-encoded equivalent of %22. Do *not* use ASCII.

Example: %25 blocks URL-encoded characters in queries.

Web Agents search for prohibited characters in query strings by comparing the characters in the query string of the URL with the ASCII values of the characters defined in the BadQueryChars parameter. For an example, see the following process;

1. The BadQueryChars parameter contains the URL-encoded value for the percent symbol (%) as shown in the following example:

   %25

2. The Web Agent receives an HTTP request that contains the following query string:

   xxx=%0d

3. The Web Agent examines the URL in the previous example, but does *not* decode the URL-encoded values. For example, the Web Agent interprets the previous example (in Step 2) as the literal string %0d, and *not* as a carriage return.
4. The Web Agent examines the values in the BadQueryChars parameter, and converts them to their ASCII values. For example, the %25 in Step 1 is converted to a percent symbol (%).
5. The Web Agent compares each character in the URL against the decoded ASCII values from the BadQueryChars parameter.
6. The Web Agent blocks the request, because the ASCII percent symbol (%) exists in both of the following places:
   - The query string of the URL.
   - The decoded (ASCII) value in the BadQueryChars parameter.

To block certain characters from query strings, set the value of the BadQueryChars parameter to include the characters you want to block.

Chapter 18: Help Prevent Attacks

This section contains the following topics:

Protect Web Sites Against Cross-Site Scripting

A Cross Site Scripting (CSS) attack can occur when the input text from the browser (typically, data from a post or data from query parameters on a URL) is displayed by an application without being filtered for characters that may form a valid, executable script when displayed at the browser.

An attack URL can be presented to unsuspecting users. When a user clicks on the URL, an application may return a display to the browser that includes the input characters, along with an error message about bad parameters on the query string. The display of these parameters at the browser can lead to an unwanted script being executed on the browser.

For example, when a user types news into a search engine web page, the application normally might return a blank field, or a response, such as:

Your search for news returned the following:

In response to an attack URL, the browser might receive a response, such as:

news<script>BadProgram</script>

The BadCSSChars parameter does not interpret the double quotation mark (") if it is entered as an ASCII character. To include the double quotation mark as a bad cross-site scripting character, enter the hexidecimal equivalent of the ASCII character, which is %22. For example:

BadCSSChars="%22"

Configure the Web Agent to Check For Cross Site-Scripting

To instruct the Web Agent to check a URL for characters that may be part of an executable script, set the CSSChecking parameter to yes. By enabling this parameter, the Web Agent scans a full URL, including the query string, for escaped and unescaped versions of the following default character set:

- left and right angle brackets (< and >)
- single quote (')

Override the Default CSS Character Set

To override the default cross-site scripting character set, enter a character set of your choice for the BadCSSChars parameter. Include the entire string of characters that you want. For example, if you set the BadCSSChars parameter to <,>, the Web Agent scans only for the left and right angle brackets.

If the Web Agent detects a problem related to the character set, it returns an Access Denied message to the user, and the logs the following message in the Agent error log:

> Caught Possible Cross Site Scripting Violation in URL. Exiting with HTTP 403 ACCESS FORBIDDEN.

Some applications require the use of the quote characters in the query string, irrespective of the web server platform. For example, some Domino applications, such as iNotes Web Access, require the use of single quotes.

To use applications that require quotes in the query string, remove quotation marks from the BadCssChars parameter.

If you leave do not use this parameter, the Web Agent checks for the default character set.

Note: For more information about cross-site scripting, go to CERT Advisory.

Safeguard Information in Cookies with HTTP-Only Attribute

To help protect against cross-site scripting attacks, you can make the Web Agent set the HTTP-Only attribute for any cookies it creates using the following parameter:

UseHTTPOnlyCookies

Instructs the Web Agent to set the HTTP-only attribute on the cookies it creates. When a Web Agent returns a cookie with this attribute to a user's browser, the contents of the cookie cannot be read by a script, even a script from the web site which originally set the cookie. This helps prevent any sensitive information in the cookie from being sent to an unauthorized third party through a script.

Default: No

To safeguard the information in cookies, set the value of the UseHTTPOnlyCookies parameter to yes.

Compare IP Addresses to Prevent Security Breaches

An unauthorized system can monitor packets, steal a cookie, and use that cookie to gain access to another system. To prevent a breach of security by an unauthorized system, you can enable or disable IP checking with persistent and/or transient cookies.

The IP checking feature enables the Web Agent to compare the IP address stored in a cookie from the last request with the IP address in the current request to see if they match. If they do not match, the Web Agent rejects the request.

The two parameters used to implement IP checking are PersistentIPCheck and TransientIPCheck. Set them as follows:

- If you enabled PersistentCookies, set PersistentIPCheck to yes.
- If you did not enable PersistentCookies, set TransientIPCheck to yes.

SiteMinder identity cookies are unaffected by IP checking.

More Information

How to Configure Single Sign-On (see page 83)
Set Persistent Cookies (see page 85)
Control Identity Cookies (see page 94)

Help Prevent DNS DOS Attacks

If an attacker sends valid HTTP requests with false IP addresses to a web server, the Web Agent would try to resolve the IP addresses to fully qualified domain names. If the volume of HTTP requests were large enough, a denial-of-service condition could affect the Web Agent and possibly the DNS servers. The following parameter controls whether the Web Agent performs DNS lookups:

DisableDNSLookups

Prevents the Web Agent from performing DNS lookups.

To help prevent DNS DOS attacks, set the value of the DisableDNSLookups parameter to yes.

Important! Fully qualified domain names must be used for cookie-based functions to work properly when the value of this parameter is set to yes.

Chapter 19: Authenticate Users with Forms

This section contains the following topics:

How Credential Collectors Process Requests

When a user requests a resource protected by a Web Agent and a credential collector, SiteMinder processes the request as follows:

Note: This process applies only to FCC, SFCC, SCC, and NTC collectors. It does not apply to the cookie provider for single sign-on.

1. A user makes a request for a protected resource.
2. The Web Agent protecting that resource contacts the Policy Server and finds out that resource is protected by a forms, an advanced SSL, or an Windows authentication scheme.
3. The Web Agent redirects the user to the appropriate credential collector, adding query data, including the target resource and its encrypted Agent name to the URL of the credential collector.
4. One of the following occurs:
   - The FCC displays the form and collects the user credentials
   - if no certificate is available, the SFCC displays the form and collects the user credentials.
   - The SCC collects the user credentials.
   - The NTC collects the user's NT credentials
5. The credential collector logs the user directly into the Policy Server. The Policy Server then creates a session.
6. The credential collector writes a session cookie to the user's browser and redirects the user back to the original Web Agent.
7. The Web Agent validates the session and permits the user access to the resource.

The r5.x, r6.x and r12 credential collectors operate differently from 4.x credential collectors. In a "mixed environment" that contains 4.x and higher Agents, you must consider how to configure an r5.x, r6.x or r12 credential collector so it can communicate with a 4.x Web Agent.

Note: For more information about SSL Authentication Schemes, see the Policy Server documentation.

Use Credential Collectors for Authentication and Single Sign-On

A SiteMinder credential collector is an application within the Web Agent that gathers specific user credentials to authenticate a user. The credentials gathered by the credential collector are based on the type of authentication scheme configured for a particular group of protected resources. Credential collectors are used for forms, SSL, and Windows authentication schemes, and for single sign-on across multiple cookie domains.

The following types credential collectors are available:

Forms Credential Collector (FCC)

Gathers credentials based on HTML forms that are presented to the user during an authentication challenge. The forms that the FCC presents are based on templates that have the file extension .fcc. For example, the Web Agent is installed with a form called login.fcc, which you can customize and use for login purposes. This file is written using standard HTML tags and some proprietary notation required by SiteMinder.

Note: When using FCC-based authentication, if a form is presented with empty credentials, a framework Web Agent does not process the request and redirects it back to the originally requested URL, hence causing the framework Web Agent not to send any communication to the policy server. In the case of a traditional Web Agent, the request is processed and sent to the policy server, which then generates an OnAuthAttempt event.

SSL Credential Collector (SCC)

Collects SSL-based credentials (credentials required by SSL-based authentication schemes) such as Basic over SSL or X509 Cert and Basic.

Note: The SCC does not handle X509 Cert and Forms or X509 Cert or Forms. X509 Cert and Forms is handled by the FCC and X509 Cert or Forms is handled by the SFCC.

Cookie Provider (CCC)

Tracks SiteMinder sessions across multiple cookie domains for single sign-on. Unlike other types of credential collectors, the cookie provider does not collect credentials or perform an authentication challenge of the user. The cookie provider is handling credentials; however, in this case, the session is the credential.

Default: SmMakeCookie.ccc

NTLM Credential Collector (NTC)

Gathers NT credentials for resources stored on an IIS Web server and accessed by Internet Explorer browsers. This scheme uses a Windows NT login name and password of a user in place of a challenge for credentials.

SSL Forms Credential Collector (SFCC)

Gathers credentials based on HTML forms (like the FCC) but the SFCC gathers them only for the X509 Cert or Forms authentication schemes.

The forms that the SFCC presents are based on a templates that end in the file extension .sfcc. For example, the Web Agent is installed with a form called login.sfcc, which you can customize and use as a login form.

MIME Types for Credential Collectors

Associated with each credential collector is a MIME type. The MIME type indicates which collector presents the authentication challenge when a user requests a resource. The following table shows each type.

Credential Collector	MIME Type
Forms Credential Collector	.fcc
SSL Credential Collector	.scc
Cookie Provider	.ccc
NTLM Credential Collector	.ntc
SSL Forms Credential Collector	.sfcc

When you configure an authentication scheme that uses a credential collector, or set up single sign-on across multiple cookie domains, the relevant MIME type is used as a file extension for a file referenced by the authentication scheme or single-sign-on configuration, for example:

- When configuring single sign-on across multiple cookie domains, you enter a URL like the following to identify the cookie provider:

  http://myserver.company.com:80/siteminderagent/SmMakeCookie.ccc

  SmMakeCookie.ccc is the default cookie provider name. You can use this name or create a name of your own; however, it must have the .ccc extension to initiate single sign-on.

- For Windows authentication, the default target file to enable this scheme is:

  /siteminderagent/ntlm/creds.ntc

  Again, you must use a file with the correct MIME type as the extension.

The FCC and SFCC are the only credential collectors that require actual files to exist on the web server where the Agent is installed. These collectors are for forms-based authentication schemes. The .fcc and .sfcc templates are required to define the HTML form presented to the user.

Configure MIME Types for Each Credential Collector

When you install a Web Agent that will act as a credential collector, there are some configuration procedures to follow so the collector can operate. The procedures differ by web server platform, not by operating system.

More information:

How to Configure Single Sign-On (see page 83)

Set Up Credential Collectors for IIS and Domino Web Servers

For IIS and Domino web servers, you have to set-up the various MIME types, represented as file extension parameters, in the Web Agent configuration file.

Map the specific MIME types for use with each credential collector. We recommend using the default values in the following table:

Agent Configuration Parameter	Credential Collector	MIME Type
CCCExt	Cookie Provider	.ccc
FCCExt	Forms Credential Collector	.fcc

Agent Configuration Parameter	Credential Collector	MIME Type
SCCExt	SSL Credential Collector	.scc
SFCCExt	SSL Forms Credential Collector	.sfcc
NTCExt	NTLM Credential Collector	.ntc

Note: Be sure to uncomment the parameter in the file.

If you do not want to use the default extensions or the defaults are already in use for other purposes, enter your own extensions and the Web Agent will honor them. For example, if you set FCCExt to .myext for the FCC, and rename the FCC template to use this extension, for example, login.myext, the Web Agent will recognize URLs ending in .myext as forms authentication requests.

Set Up Credential Collectors for Sun Java System Web Servers

For Sun Java System web servers on Windows or UNIX platforms, the MIME types are set up automatically when you install the Web Agent. There is no other configuration that is necessary.

Set Up Credential Collectors for Apache Web Servers

For Apache web servers on Windows or UNIX, you have to modify the httpd.conf file *after* you install the Web Agent.

Specifically, you need to add entries to the Alias section that direct the web server to the installed location of the Web Agent and the Web Agent's samples directory, where the forms templates reside. You also have to add entries to the AddHandler section for each MIME type.

Note: For more information, see the SiteMinder Web Agent Installation documentation.

Configure Credential Collectors in a Mixed Environment

Beginning with 5.x QMR 2, the forms (FCC/SFCC), SSL (SCCs), and NTLM (NTC) credential collectors operate differently than 4.x credential collectors. When a user submits credentials, the credential collector does not have to create a credential cookie in the user's browser and send the user back to the original Web Agent. Instead, the credential collector can log the user in to the Policy Server directly on behalf of the Web Agent protecting the requested resource.

Note: We recommend using credential collectors to log users in directly rather than setting cookies. Using credential collectors to log users in better secures user credentials because these credentials are not being passed around the network in cookies. This is an important consideration when you are configuring credential collectors in a mixed environment.

For a credential collector to log a user in, it needs the name of the Web Agent protecting the requested resource and the credentials supplied by the user.

To learn the Agent name, a credential collector uses the following process:

1. Use the SMAGENTNAME query parameter that the original Web Agent adds to the query string of the URL as it redirects the user to the credential collector.
2. If there is no Agent name appended to the URL, use the Agent name from Agent name-to-host name mappings in the credential collector's Agent configuration file or Agent Configuration Object.

   Each mapping specifies the name and IP address of a host using that collector for its protected resources. Mappings are defined in the AgentName parameter.
3. If no Agent name mappings are configured, use the fully qualified host name of the target URL as the Agent name. This is determined by enabling the AgentNamesAreFQHostNames parameter in the Agent's configuration.

   This parameter is disabled by default, so the credential collector uses the value of the DefaultAgentName parameter as the Agent name.

When you upgrade, you must consider the credential collector algorithm and how it affects the configuration of an FCC, SCC, SFCC, or NTC so it can communicate with a 4.x Web Agent.

Use FCCs and NTCs in a Mixed Environment

To process requests, the FCC and NTC rely on the user credentials and the name of the Web Agent that is protecting the requested resource. However, 4.x Web Agents and third party Agents who may post to the FCC and NTC do not pass the Agent name on the URL they send.

The following configuration options help FCCs and NTCs to operate with 4.x Web Agents:

- Use Compatibility Mode—to enable a r5.x, r6.x, or r12.0 SP2 FCC/NTC to serve up forms for resources protected by 4.x Web Agents or third party applications, enable the FCCCompatMode parameter. Traditional Web Agents have the FCCCompatMode parameter is enabled by default; framework Agents have the FCCCompatMode parameter is disabled by default.

  Enabling this parameter makes a r5.x, r6.x, or r12.0 SP2 Agent handle forms and NTLM credential collection like a 4.x Agent, which means that a form or NTLM credential cookie is written to a user's browser and the user is redirected back to the Agent before she can log in at the Policy Server. This permits the Web Agents to interoperate.

  When the value of the FCCCompatMode parameter is set to no, compatibility withFCCCompatMode 4.x Agents is disabled. In an exclusively r5.x, r6.x, or r12.0 SP2 environment, set the value of the parameter to no.

  Important! Setting this parameter to no removes support for version 4.x of the Netscape browser.

- Specify Agent name mappings—FCC only: If you disable backward compatibility (by setting the value of the FCCCompatMode parameter to no), map the AgentName parameter to the name and IP address of each host using that FCC for its protected resources. Set up these mappings in the FCC's Agent configuration file or its Agent Configuration Object on the Policy Server.

  Example mappings:

  myagent, 123.1.12.1

  myagent, www.sitea.com

- Use Host Names as Agent Names—FCC only**:** If the first two options in the algorithm are not optimal, you can set the value of the AgentNamesAreFQHostNames parameter to yes. This tells the FCC to use the fully qualified host name in the target URL as the Agent name. For example, if the URL string includes:

  url?A=1&Target=http://www.nete.com/index.html

  The www.nete.com portion of the Target string serves as the Agent name.

  By default, this parameter is set to no. Consequently, the value of the DefaultAgentName parameter is used as the Agent name.

The following tables list guidelines for configuring r5.x, r6.x, or r12.0 SP2 and 4.x FCCs and NTCs, and describes how each behaves in a mixed environment:

Notes:

- Beginning with Web Agent 5.0, NTLM credential collectors can redirect users from non-IIS Web Servers to IIS Web Servers.
- For framework Web Agents, refer only to the instructions where FCC compatibility mode is disabled.

Web Agent Protecting Resources	r5.x, r6.x, or r12.0 SP2 FCC in FCC Compatibility Mode	r5.x, r6.x, or r12.0 SP2 FCC - FCC Compatibility Mode Disabled
r5.x, r6.x, or r12.0 SP2	■ FCC issues a credential cookie. ■ Certificate **and** Forms authentication will not work. ■ Certificate **or** Forms authentication will not work.	■ FCC issues a session cookie ■ Certificate **and** Forms authentication works. ■ Certificate **or** Forms authentication works.

Web Agent Protecting Resources	4.x QMR 2/3/4 FCC
4.x QMR 5 or 4.x QMR 6	■ Agent issues a credential cookie ■ Certificate and Forms authentication will not work ■ Certificate or Forms authentication works
r5.x, r6.x, or r12.0 SP2	■ Agent issues a credential cookie ■ Certificate and Forms authentication will not work ■ Certificate or Forms authentication works

Note: For more information about SSL Authentication Schemes, see the Policy Server documentation.

Web Agent Protecting Resources	**r5.x, r6.x, or r12.0 SP2 FCC in FCC Compatibility Mode**	**r5.x, r6.x, or r12.0 SP2 FCC - FCC Compatibility Mode Disabled**
4.x QMR 5 or 4.x QMR 6	■ NTC issues a credential cookie.	■ NTC issues a session cookie
r5.x, r6.x, or r12.0 SP2	■ NTC issues a credential cookie.	■ NTC issues a session cookie
Web Agent Protecting Resources	**4.x QMR 2/3/4 NTC**	
4.x QMR 5, 4.x QMR 6	■ Agent issues a credential cookie	
r5.x, r6.x, or r12.0 SP2	■ Agent issues a credential cookie	

Use SCCs in a Mixed Environment

To enable 4.x Web Agents and r5.x, r6.x, or r12.0 SP2 SCCs to interoperate, do one of the following:

- Specify Agent name mappings: Map the AgentName parameter to the host name and IP address of each host using that SCC for its protected resources. You set up these mappings in the SCC's Agent configuration file or its Agent Configuration Object at the Policy Server.
- Use Host Names as Agent Names: If you do not specify Agent name mappings, you can set the AgentNamesAreFQHostNames parameter to Yes. This tells the SCC to use the fully qualified host name in the target URL as the Agent name.

  For example, if the URL string is:

  url?A=1&Target=http://www.nete.com/index.html

  The www.nete.com portion of the Target string serves as the Agent name.

  By default, this parameter is set to no. Consequently, the value of the DefaultAgentName parameter is used as the Agent name.

The following table shows how 4.x and r5.x, r6.x, or r12.0 SP2 Agents acting as SCCs operate in a mixed environment:

Web Agent Version	4.x QMR 2/3/4 SCC	r5.x, r6.x, or r12.0 SP2 SCC
4.x QMR 5 or 4.x QMR 6	■ Agent issues an SSL credential cookie. ■ Certificates cannot be collected without redirecting requests, even if the original connection from the browser to web server is over SSL.	■ Create mappings in AgentName parameter or set AgentNamesAreFQHost Names to Yes. ■ SCC issues a session cookie ■ Certificates cannot be collected without redirecting requests, even if the original connection from the browser to web server is over SSL.
r5.x, r6.x, or r12.0 SP2	■ Agent issues an SSL credential cookie. ■ Certificates can be collected without redirecting requests.	■ SCC issues a session cookie ■ Certificates can be collected without redirecting requests.

Note: For more information about SSL Authentication Schemes, see the Policy Server documentation.

Configure the FCC to Use a Single Resource Target

You can configure the FCC to direct users to a single resource by hard coding the target in the login.fcc template file.

To configure the FCC to use a single resource target

1. Open the login.fcc file, located in *agent_home*/Samples.
2. Add @target=*target_resource* to the FCC.
3. Add the following entry:

   @smagentname=*agent_name_protecting_resource*

   For example: @smagentname=mywebagent
4. Set the EncryptAgentName parameter to no. This is required because no method exists to encrypt the Agent name after you hard code it in the file.
5. Set the EncryptAgentName to no for any other Agent using this FCC.

Note: For more information, see the Policy Server documentation.

Enable Forms Cache to Improve Performance

The form cache is a single repository for storing form template data. Storing template data in cache avoids the Agent's repetitive reading of .fcc files for the same data. Consequently, when processing forms authentication requests, the form cache can improve the Agent's ability to fetch data that FCC threads use.

When a resource with an FCC extension is accessed, the corresponding template file is read and processed by the FCC. A Web Agent takes hundreds of these requests each second.

The form cache relieves the FCC by storing form template files in memory where they can be read easily. Virtual memory access is faster than disk access, allowing FCC components to process forms more quickly with reduced strain on the host server. The improved processing time increases the FCC's capacity for serving requests for each web server and makes forms authentication more efficient.

Data Stored in the Form Cache

The data stored in form cache consists of the form template text, pre-parsed into data structures that make future processing by the FCC more efficient.

These data structures include:

- Form locale data for internationalization
- An ordered list of data objects containing raw text in UTF-8 format, template directive information and function/variable information for substitution from the request environment.

By pre-processing the raw text from the FCC file before it is stored in cache, the FCC can reduce redundant operations and process forms more efficiently. The order of the data objects must be maintained to comply with existing FCC functional design. By design, directives, functions and variables are always processed from the top of the FCC file down. The FCC understands how to process the various ordered lists to properly support any given FCC template's intended functionality.

Configure the Form Cache

Forms can be cached to improve performance and reduce unnecessary network traffic. You can control the settings of form cache with the following parameters:

EnableFormCache

Controls the forms template cache. Setting this parameter to yes, improves the performance of forms authentication. To disable the cache, set this parameter to no.

Default: Yes

FormCacheTimeout

Specifies the number of seconds that an object may reside in cache before being considered invalid. When the timeout interval expires, the date and time of the form template file is compared against the time that the cache object was created. If the object in cache is stored more recently than the file on the system's disk, the timeout is reset for another interval; otherwise, the object is removed from cache.

Default: 600

To configure the forms cache

1. Set the value of the EnableFormCache parameter to yes.
2. If you want to change the timeout interval for the form cache, set the value of the FormCacheTimeout value to the number of seconds you want.

   The form cache is configured.

Disable FCC Realm Context Confirmation to Improve Performance

During forms authentication, the Web Agent makes an IsProtected call to the Policy Server to determine if the requested resource is protected. After this first call, the Web Agent typically makes an additional IsProtected call to the Policy Server. This second call establishes a realm context so that the Web Agent can log a user in with an FCC to access a protected resource. You can control whether the Web Agent makes this additional call using the following parameter:

FCCForceIsProtected

Specifies whether the Web Agent makes an additional IsProtected call to the Policy Server to establish a realm context so that the Web Agent can log a user in to access a protected resource.

When this parameter is set to no, the Web Agent uses the realm information obtained from its initial IsProtected call to the Policy Server instead.

Default: Yes

To improve performance by disabling the FCC realm context confirmation, set the value of the FCCForceIsProtected parameter to no.

Use a Relative Target for Credential Collector Redirects

You can instruct the Web Agent to use a relative URI instead of an fully qualified URL when directing requests to a credential collector and target resource. Using a relative URI prevents requests from being processed by credential collectors on other systems installed with Web Agents.

Note: This setting applies to all credential collectors *except* the cookie credential collector (CCC). The CCC must use a fully-qualified domain name for this parameter. OnAuthAccept responses will not work properly with a CCC if a relative URI is used.

Typically, a fully qualified URL is appended to the credential collector URL. For example:

url?A=1&Target=http://www.nete.com/index.html.

To use only a relative URI, set the TargetAsRelativeURI parameter to yes. If set to yes, the target parameter appended to the credential collector URL is a relative target, such as url?A=1&Target=/index.html. In turn, when the credential collector redirects back to the Web Agent protecting the target resource, it is a relative redirect. Also, the Web Agent rejects any target that does not begin with a forward slash (/).

The default value for this parameter is no, so a fully qualified URL is always used.

Define Valid Target Domains for CCC Processing

Web Agents can help protect from phishing attempts that could redirect users to a hostile web site, with the following parameter:

ValidTargetDomain

Specifies the domains to which a credential collector is allowed to redirect users. If the domain in the URL does not match the domains set in this parameter, the redirect is denied.

Default: No default

This parameter is supported by all advanced authentication schemes, including forms credential collectors (FCCs).

During processing, the ValidTargetDomain parameter identifies the valid domains for the target. Before redirecting the user, the Web Agent compares the values in the redirect URL against the domains in this parameter. Without this parameter, the Web Agent redirects the user to targets in any domain.

The ValidTargetDomain parameter can include multiple values, one for each valid domain.

For local Web Agent configurations, specify an entry, one on each line, for each domain, for example:

validtargetdomain=".xyzcompany.com"

validtargetdomain=".abccompany.com"

Enable FCCs/SCCs to Use Agent Names as Fully Qualified Host Names

The AgentNamesAreFQHostNames parameter enables forms and SSL credential collectors to use the fully qualified host name of the target URL as a Web Agent name. For example, if the URL string includes:

url?A=1&Target=http://www.nete.com/index.html

The www.nete.com portion of the Target string serves as the Web Agent name.

The credential collector uses this parameter if:

- No Agent name is appended to the URL from the target Web Agent.

  (This may be the case with third party Agents.)

- You have not configured Agent-to-host name mappings in the AgentName parameter.

If AgentNamesAreFQHostNames is set to no, the credential collector uses the value of the DefaultAgentName parameter as the name of the target Web Agent.

Map Agent Identities and Web Servers for Use By FCCs and SCCs

The AgentName parameter and its associated IP addresses provide mappings between web server interfaces and Agent names defined in the policy store. Web Agents need to make Agent API calls in the proper Agent name context for the correct set of rules and policies to apply.

If the Web Agent is acting as a forms or SSL credential collector, it needs the name of the Web Agent protecting the requested resource and a user's credentials to process requests. By default, SiteMinder includes the Agent name in the URL that redirects the user from the Web Agent to the credential collector. However, if you are working with other applications, you can map the AgentName parameter to the name and IP address of each host using the collector. The credential collector uses the mappings to resolve the Agent name.

The following are example values for the AgentName parameter:

- myagent1,123.1.1.12
- myagent, www.sitea.com

Preserve Data Posted to a Form

Beginning with SiteMinder 5.0, data that a user enters in a form is preserved, even if a timeout or other interruption occurs during the posting. This is called POST preservation. After the user enters the credentials and submits the form, the preserved data is re-posted to the application.

Modify the .fcc File for Forms POST Preservation

You may have forms that obtain credentials from SiteMinder releases prior to v5.0. To preserve data during a POST, add the following line to the .fcc form template that is presented when a user tries to access the resource:

```
<input type=hidden name=postpreservationdata
value="$$postpreservationdata$$">
```

Note: POST preservation is *not* supported with CGI and Perl based Password Services, ACE authentication, or any custom authentication scheme that posts to an FCC.

Enable Post Preservation between Framework and Traditional Agents

Framework Agents handle POST preservation data differently than Traditional Agents do. If your SiteMinder environment uses a combination of Framework and Traditional agents, and resources hosted by one type of Agent are protected by Forms Credential Collectors (FCCs) hosted on the other type of agent, you must specify the proper template file with the following parameter:

PostPreservationFile

Enables the transfer of POST preservation data between Traditional and Framework Agents by specifying the path to *one* of the following POST-preservation-template files:

- tr2fw.pptemplate—Indicates that resources hosted on a server running a traditional agent are protected by an FCC running on a Framework agent.
- fw2tr.pptemplate—Indicates that resources hosted on a server running a Framework agent are protected by an FCC running on a Traditional agent.

Default: No default

Example: *web_agent_home*/samples/forms/fw2tr.pptemplate

To enable post preservation between Framework and Traditional agents

1. Determine which resources are protected by FCCs running on a different type of Agent.
   a. Create a list of Traditional Agents hosting resources that are protected by FCCs running on Framework Agents.
   b. Create a list of Framework Agents hosting resources that are protected by FCCs running on Traditional Agents.
2. For any traditional Agents hosting resources (those you listed previously in step 1a), set the value of the PostPreservationFile parameter to the path of the tr2fw.pptemplate file.
3. For any Framework Agents hosting resources (those you listed previously in step 1b), set the value of the PostPreservationFile parameter to the path of the fw2tr.pptemplate file.

4. For all of your Framework Web Agents that communicate with Traditional Agents, set the value of the following parameter to yes:

   LegacyPostPreservationEncoding

   Specifies whether the Web Agent encodes any POST preservation data in a way that is compatible with the older, Traditional, Web Agents, or with the newer, Framework Web Agents. When the value of this parameter is set to yes, the encoding is compatible with the Traditional Web Agents. When the value of this parameter is set to no, the encoding is compatible *only* with the Framework Web Agents.

   Default: No

5. Restart the web servers hosting your resources.

   POST preservation is between Framework and Traditional agents is enabled.

More information:

Types of Web Agents (Traditional and Framework) (see page 23)

Disable POST Preservation

If you do not need to use POST preservation, you may disable it with the following parameter:

PreservePostData

Specifies whether the Web Agent preserves POST data when redirecting requests. When the user is challenged for advanced authentication, such as forms or certificate authentication, the post data is preserved during the authentication phase.

Default: Yes

To disable POST preservation, set the value of the PreservePostData parameter to no.

Use the safeword.fcc File for SafeWord Forms Authentication

The Policy Server can authenticate users against a SafeWord authentication server, including users who are logging in via SafeWord hardware tokens.

One of the prerequisites for using the SafeWord forms-based authentication scheme is to have a customized safeword.fcc file residing on a web server where the SiteMinder Web Agent is installed. This web server must be in the cookie domain in which you implement HTML Forms authentication.

The safeword.fcc file defines the forms that a user sees during SafeWord authentication. Depending on the value of the authentication code sent by the Policy Server to the credential collector, the form that the user is asked to fill out changes. In the safeword.fcc file you can see the different text for each authentication code, as indicated by the directive smauthreason.

To customize the safeword.fcc file for your enterprise, you can modify the HTML layout of the form but not the type of credentials that the user must provide for a particular form. You may also want to modify the form logo. The file uses ISO-8859-1 encoding.

The sample safeword.fcc file is located in the directory:

web_agent_home/Samples/Forms

Note: For more information, see the Policy Server documentation.

Use a Special Forms Template for Passport Authentication

Beginning with Web Agent 5.x QMR1, n FCC file named loginusername.fcc was provided for use with the Passport authentication scheme. If you configure SiteMinder to use this form, when a user requests a protected resource, SiteMinder will:

1. Recognize a signed-in Passport user as a mapped user from the SiteMinder user directory.
2. Present the form, which:
   - Automatically displays the user name of the mapped user
   - Prompts for the corresponding password

To use the loginusername.fcc file:

1. Edit the value of the IgnoreExt Web Agent parameter by removing the .fcc entry from the list of extensions that the Agent should ignore.
2. Protect loginusername.fcc, using the Passport (Custom) authentication scheme.

   Note: For more information, see the Policy Server documentation.

3. For each realm protected by the Passport authentication scheme, create a response on the Policy Server. For each response, configure a Web Agent response attribute as follows:
   a. Select WebAgent-HTTP-Header-Variable from the Attribute drop-down list.
   b. Select the User Attribute radio button from the Attribute Kind group box.
   c. In the Attribute Name field, enter the name of the user directory attribute that corresponds to the user name or user id. For example, if an LDAP directory contains the users mapped to Passport holders, enter uid.
   d. In the Variable Name field, enter a name for the response variable, such LDAPUID.

      Note: For more information, see the Policy Server documentation.
4. Edit the loginusername.fcc form to reflect the Variable Name value. Continuing with this example, the variable name is LDAPUID.

You can add these advanced features to the Agent configuration file or an Agent Configuration Object.

How to use Forms with ACE Authentication

If you want to use a SiteMinder form together with an ACE authentication scheme, do one of the following:

- If you are creating a new authentication scheme, use the following form as a template:

  web_agent_home\samples_default\forms\smaceauth.fcc

- If you are modifying an existing authentication scheme to use ACE, then add the following directive below the other directives in the forms.fcc file:

  @smacefcc=1

Chapter 20: Manage Password Services

This section contains the following topics:

Supported Approaches for Using Password Services with Web Agents

To support password services, you can use either of the following approaches:

- FCC-based approach (required for POST preservation)
- CGI-based approach (deprecated)

Note: POST preservation is *not* supported with CGI and Perl based Password Services, ACE authentication, or any custom authentication scheme that posts to an FCC.

We recommend the FCC-based approach. CGI-based password services is deprecated, which means it will not be supported at a future release.

FCC Password Services and URL Query Encryption

The FCC Password Services application enables query data on the URL to be encrypted, further securing Agent interactions. You can only encrypt query data with FCC Password Services. FCC Password Services files include:

- smpwservices.fcc

  This FCC is installed with the Web Agent and is located at:

  web_agent_home/samples/forms

  If Password Services is invoked and there is no password policy configured, the SiteMinder Administrator at the Policy Server should set the environment variable NETE_PWSERVICES_REDIRECT to a relative path for smpwservices.fcc.

  The path is:

  /siteminderagent/forms/smpwservices.fcc

  The new FCC displays the Password Services form based on the FCC directives authreason and username.

- smpwservices.unauth

  This file handles errors that occur during GET/POST actions of the Password Services forms.

  This file is similar to other FCC unauthorized files that are invoked if there is a failure processing the request during the POST. This FCC handles error conditions, such as an empty TARGET variable. The error reporting is intended to be synchronized with the CGI-based Password Services and for handling any other unknown errors caused by an FCC POST.

- smpwservicesUS-EN.properties

  This properties file is used by smpwservices.fcc to display the user-friendly messages on the Password Services forms.

  This properties file has the user-friendly messages, which an administrator can modify depending on what he wants to display on the Password Services forms. The format for the message is name=value.

Configure FCC Password Services

FCC Password Services is configured much like the CGI or JSP versions—a SiteMinder Administrator needs to configure password policies that are associated with a user directory or namespace. However, the Redirection URL field that you are required to configure must be set to a relative path for the smpwservices.fcc. This path is:

/siteminderagent/forms/smpwservices.fcc

The redirection path ensures that FCC Password Services works correctly.

Note: For more information, see the Policy Server documentation.

How to Enable User-Initiated Password Changes with FCCs

You can configure the FCC Password services features of SiteMinder to allow users to change their own passwords whenever they want. To enable user-initiated password changes with FCCs, use the following process:

1. Confirm that your user directory contains attributes that support Password Policies.
2. Use the Administrative UI to do the following tasks:
   a. Create an FCC-based password policy and protect the resources you want.
   b. Configure the password policy to allow authorized users to change their passwords.
3. Create a password change URL in your FCC form that includes the following parts:
   - The FQDN of the logon server (example: http:*logonserver.example.com)*
   - The URI of the FCC based Password services (example: siteminderagent/forms/smpwservices.fcc?
   - The name of the SiteMinder Web Agent (SMAGENTNAME)
   - A target resource that is protected by SiteMinder (TARGET).
4. Embed the URL as a link in one or more unprotected web pages, as shown in the following example:

   ```
   <a href="http:logonserver.example.com/siteminderagent/forms/smpwservices.fcc?SMAUTHREASON=34&SMAGENTNAME=$$smencode(smagentname)$$&TARGET=$$smencode(target)$$">Change Password</font></a>
   ```

5. Test the password change function with the following steps:
   a. Display a web page that has the password change link you created in Step 3.
   b. Click the password change link.

      The password change form appears.
   c. Fill out the password change form and submit it.

      If the password change is successful, a confirmation page appears, and shows a link to the protected target resource.
   d. Click the link and ensure that the resource appears.
   e. Close and re-open your browser. Try to access the protected resource using your new password.

      If you can access the resource with your new password, the password change is successful.

Configure SecureID Authentication with FCC Password Services

You must modify the SecureID HTML Form template using the Administrative UI if you are using SecureID as your authentication scheme and both of the following conditions exist in your environment:

- The FCC Password Services feature is configured
- The value of the SecureUrls parameter for the Web Agent is set to yes

SecureID is implemented using Password Services, which is why you must modify the authentication scheme's template.

To configure SecureID Authentication with FCC password services, add the path to the smpwservices.fcc file in the Target field of the SecureID template, as shown in the following example:

```
/siteminderagent/forms/smpwservices.fcc
```

Localize FCC-based Password Services Change Forms

You can localize the user messages for FCC-based Password Services. The following process for modifying FCC-based Password Services works for any locale:

1. Create a new FCC folder on the web server for a new locale or use an existing folder if appropriate for your locale. The naming convention for the folder is forms*locale*.
2. Place a copy of the relevant Password Services files in the new folder.
3. Modify the files to accommodate the locale, such as changing the English messages to the language for your locale. Repeat this step for all of the files that you will use in the locale.
4. In the Administrative UI, change the value of the Redirection URL field in the Password Policy.

   For example, to use FCC Password Services for Japanese users, put a copy of the following files in the folder formsja, located in *web_agent_home*/samples:

   - smpwservices.fcc, located in *web_agent_home*/samples/forms
   - smpwservices.unauth, located in *web_agent_home*/samples/forms
   - A new properties file, smpwservicesja.properties

Localize CGI-based Password Services Change Forms

For Password Services change forms, the language encoding is read from the Web browser. The Password Services CGI reads the ACCEPT_LANGUAGE variable from the Web browser to determine the language in which to display the password change forms.

For each language, the change forms use a different properties file. The properties file defines certain aspects of the text displayed on the HTML pages.

The following properties files are automatically installed with the Web Agent:

- English: PasswordServicesUS-EN.properties
- French: PasswordServicesFR-FR.properties
- Japanese: PasswordServicesJP-JP.properties

If the ACCEPT_LANGUAGE variable indicates a language for which no properties file exists, the English properties file is used.

To create and use a properties file for another language

1. Translate one of the properties files provided by the SiteMinder Web Agent into the language of your choice. The files are located in *web_agent_home*/pw.

   For example, copy the PasswordServicesUS-EN.properties file and translate anything to the right of the equals sign (=).

2. Rename the file according to the convention for Password Services, which is PasswordServices*language-country*.properties

   Note: RFC 3066 defines the language tags that you can use as part of the naming convention. See IETF. Also, if the ACCEPT_LANGUAGE variable returned by the browser uses only the language tag as its value, then rename the file as PasswordServices*language.*properties instead.

   Be sure the browser is set for the correct language.

   If you set multiple languages for the browser, Password Services will only use the first language in the list. If the Password Services CGI cannot find a properties file for that language, the English properties file is used. The other languages specified for the browser are ignored.

Use a Fully Qualified URL for Password Services Redirects

You can make the Web Agent use a fully-qualified URL for redirects to the Password Services application by setting the following parameter:

ConstructFullPwsvcUrl

Instructs the Web Agent to generate a URL with a fully qualified domain name for redirecting users to the Password Services application. This lets you host the Password Services application on a particular web server. The Web Agent generates a URL that resembles the following example:

HTTP://my.server.com:80/path/to/passwordservices.cgi

If a fully-qualified URL is *not* used, the Web Agent assumes that the Password Services application is hosted on the same web server and uses a relative URL for redirects.

Default: No

For example, if you have all of your password services on a specific web server, you can set this parameter to yes. If this parameter is set to no, any customized content used with password services would have to be copied to every web server.

To use a fully-qualified URL for Password Services redirects, set the value of the ConstructFullPwsvcUrl parameter to yes.

DMS2 (Registration Services) and Localization

SiteMinder passes localized settings to resources differently depending on whether the resource is protected or unprotected.

Pass on Localized Settings to Protected Resources

To pass localized settings to a protected resource, use the SiteMinder static response HTTP-Header attribute, configured using the Administrative UI. The variable name of the static attribute is SM_LOCALE and the value will be set to *language-country*, for example, FR-FR.

The resource is localized based on the SM_LOCALE SiteMinder header variable. If a response is not set for SM_LOCALE, the default language is "US-EN".

Note: The default, "US-EN," does not follow the *language-country* pattern for backwards compatibility with an early version of the Password Services CGI.

Pass on Localized Settings to Unprotected Resources

For an unprotected resource like self-registration, the SM_LOCALE value is set as a hidden variable upon submitting a DMS form. For example:

```
<FORM NAME="newForm" METHOD=POST>
<INPUT TYPE=HIDDEN NAME="SMLOCALE" value="EN-US">
```

To ensure localized settings are passed correctly:

1. Remove the SM_LOCALE variable from the resource target in both the loginandregister-dms-i.fcc and loginandregisterwithforgottenpassword-dms-i.fcc files.

   These files are located in *web_agent_home*/Samples/dmsforms.

2. Verify that the locale value is captured through the header variable (SM_LOCALE) for protected resources and as a query parameter (SMLOCALE) for unprotected resources.

   Note: For more information, see the Policy Server documentation.

3. For an unprotected resource, verify that the locale value is passed from one form to another.

Note: If you are using Registration Services (DMS2) together with Password Services, note that Password Services no longer uses SM_LOCALE to determine localized settings. Instead, it uses the ACCEPT_LANGUAGE variable from the user's browser. Although DMS2 still uses SM_LOCALE when it passes the value of SM_LOCALE to Password Services, this value is disregarded in favor of the ACCEPT_LANGUAGE variable.

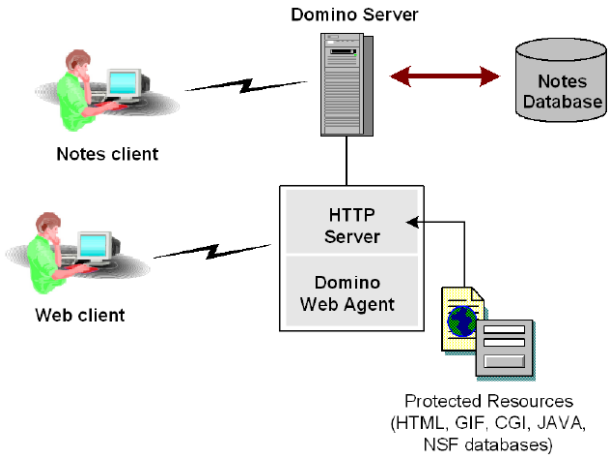
Chapter 21: Domino Web Agents

This section contains the following topics:

Domino Agents Overview

The Domino Application Server is a messaging and Web application platform that offers secure access for Lotus Notes clients. The Domino Web Agent protects only the HTTP interface of the Domino Application Server, controlling access to HTML, JAVA, CGI, and other Web resources, such as Notes served over the web. It does not protect the Notes server.

The following illustration shows how the Domino Web Agent integrates with the Domino server.



Domino stores data in groups of Notes databases. Resources in a Notes database can be a variety of objects, such as documents, views, forms, and navigators. These objects can include text, video, graphics, and audio content.

Notes objects are opened using a URL. To make Notes objects available for the Web, Domino dynamically creates Web pages from the objects in the Notes database. In the case of database views, Domino also creates URL links to the documents in a view. The dynamic creation of pages from the Notes database provides users with the most current information.

Domino URL Commands

A user's access over the Internet to resources on a Domino server is based on the URL. Therefore, understanding the Domino URL syntax is important.

Though Domino can interpret standard URLs, such as:

http://www.mycompany.com/index.html

Domino URL commands can also have the following syntax:

http://host/database.nsf/Domino_object?Action_Argument

host

Indicates the DNS entry or IP address of the server.

database

Specifies the database file name with the path relative to notes \data directory or the database Replica ID.

Domino_object

Specifies the object in the database, for example, a view, document, form, or navigator.

Action

Identifies the operation performed on the Notes object. For example: ?OpenDatabase, ?OpenView, ?OpenDocument, ?OpenForm, ?ReadForm, ?EditDocument. If there is no action specified in the URL, the default is used.

Default: ?Open.

Argument

Provides further definition of how an object is delivered by the Domino server. For example, if the action and argument is ?OpenView&Expand=5, this argument specifies the number of rows in a view to be shown in an expanded format.

The following is an example of a URL to access a view in a Notes database called financials.nsf:

http://www.anysite.com/financials.nsf/reports?OpenView

Domino Aliases

One of the Notes database conventions is to create aliases for objects. For example, the alias might identify a resource by its Notes ID or Replica ID instead of the object name. Using aliases makes programming easier for developers because the names of the Notes resources can change without requiring code changes.

The following Domino URLs access the same resource though the resource is identified by its aliases:

- http://www.domino.com/85255e01001356a8852554c20756?OpenView
- http://www.domino.com/85267E00075A80C/people?OpenView
- http://www.domino.com/__852567E00075A80C.nsf/people?OpenView

Regardless of how a resource is identified, the Domino Web Agent converts all Domino naming conventions into a standard URL based on the name of the database resource. This simplifies data entry into the SiteMinder policy store.

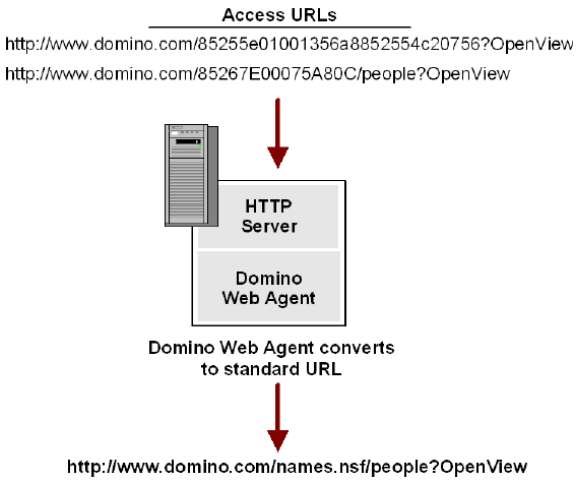
For example, the following Domino URLs are pointing to the people view in the names.nsf database. The database and view are referred to by Replica ID and Notes ID:

- http://www.domino.com/85255e01001356a8852554c20756?OpenView
- http://www.domino.com/85267E00075A80C/people?OpenView

The Domino Web Agent converts these URLs to a standard URL, as follows:

- http://www.domino.com/names.nsf/people?OpenView

The following illustration shows the conversion of aliases to a named object.



Convert Notes Document Names

Unlike views and forms, Notes documents do not have names; they are saved to the database with a reference to the form that was used to create the document. If a user is trying to access a document and the Domino Web Agent cannot convert it to a readable name, the Agent uses the name of the form that generated the document to create a URL. This applies only to documents. If there is no original form, the Agent uses the embedded form. If neither apply, the document is protected using the Domino identifier $defaultForm.

For example, if the incoming URL is:

http://www.domino.com/names.nsf/8567489d60034we50938450098?OpenDocument

The Agent uses:

http://www.domino.com/names.nsf/Person?ReadForm

In this example, Person is the name of the document.

Configure the Domino Web Agent

The Domino Web Agent uses all the standard Web Agent settings to do the following:

- Configure the Web Agent to communicate with the Policy Server
- Add and remove Agent identities for virtual servers
- Modify Web Agent settings
- Configure single sign-on
- Configure error message logging

You can configure these centrally at the Policy Server or locally in the Agent configuration file.

In addition to the standard functions, there are Domino-specific parameters you can set.

More Information

Configure Domino-Specific Agent Functions (see page 262)

Configure Domino-Specific Agent Functions

In addition to the standard Web Agent settings, there are specific Domino configuration parameters that you can set only for the Domino Web Agent. These settings determine how Domino authenticates and authorizes a user with SiteMinder. You can configure these settings centrally at the Policy Server or locally in the Agent configuration file.

Note: SiteMinder Delegated Management Services (DMS) is not supported in a Domino environment. In addition, the Domino Web Agent does not support the auditing feature used to track user activity.

Authenticate Users with the Domino Server

The Domino server must authenticate and authorize users even if SiteMinder has already gone through this process. SiteMinder works with Domino's authentication process by providing the Domino server with a user identity that is also configured in the Domino Directory, which is the list of users and their privileges. The Domino server uses this identity to authenticate and authorize the user for access to database resources.

Note: A user name must be resolved unambiguously, or else the Domino Agent denies the authentication request. This may require some adjustments in your user directory.

The Domino Web Agent identifies the user to the Domino server as one of the following:

- Super user
- Actual user
- Default user

To determine which identity the Domino Web Agent uses when communicating with the Domino server, you configure the following parameters:

SkipDominoAuth

Determines which name to pass to the domino server for server authentication.

DominoSuperUser

Identifies a user who has access to all resources on the Domino server.

DominoDefaultUser

Identifies a user with default access to the Notes database, which means this person has general access privileges.

Note: You can configure the DominoSuperUser and DominoDefaultUser locally, in the Agent configuration file, or centrally, in the Agent Configuration Object. In the Agent configuration file, these settings have encrypted values. In the Agent Configuration Object, you have the choice of encrypting these values or leaving them in plain text.

More Information

Force SiteMinder to Authenticate Users (see page 264)
Authenticate as the Domino Super User (see page 265)
Authenticate as the Actual User or the Default User (see page 265)

Force SiteMinder to Authenticate Users

To have SiteMinder (and not Domino) authenticate users, set the SkipDominoAuth parameter to yes.

With SkipDominoAuth set to yes and a Super User defined, SiteMinder first identifies and authorizes the user. The Domino Web Agent then identifies that user to the Domino Server as the Super User. As a Super User, the user has access to any resource on the Domino server, assuming the user has the appropriate ACLs.

You should also set SkipDominoAuth parameter to yes when users are not stored in the Domino Directory because Domino will not have an identity to use for authorization privileges.

If you set SkipDominoAuth to no, Domino authenticates users on its own using the actual user name or the default user name.

The following table shows how the setting of the SkipDominoAuth parameter affects how the user is identified.

SkipDominoAuth Value	Identified to the Domino Server As	Notes
yes	Super User	Super User must be defined in the Domino Directory
no	Actual User	User must be in the Domino Directory
no	Default User	User must be in the Domino Directory
no	Super User	The requested resource is automatically authorized, meaning that no authentication challenge will be presented to the user

More Information

Authenticate as the Actual User or the Default User (see page 265)

Authenticate as the Domino Super User

A Domino Super User is a user who has access to all resources on the Domino server. If your Web site or portal is designed with SiteMinder in mind, you are securing resources and applications by implementing SiteMinder policies. As a result, the Domino server does not have to restrict user access based on its own security. In this case, users can be identified as the Super User for Domino's authentication purposes.

To identify the user as the Super User, you enable the SkipDominoAuth parameter and specify a value for the DominoSuperUser parameter. This action makes sure that SiteMinder and not Domino authenticates users. The user that you specify must also be in the Domino Directory.

Authenticate as the Actual User or the Default User

If a user is defined in the Domino Directory, Domino authenticates that user with their user name. However, if the user is not in the Domino Directory, and they have been authenticated by SiteMinder against another user directory, then the Domino Web Agent identifies that user to the Domino server as the DominoDefaultUser.

The default user has default access to the Notes database, which means this person should have general access privileges such as Domino's depositor, reader, or author level of access, configured in ACLs.

For the Domino Agent to use this value, set the SkipDominoAuth parameter to no.

There may be some Notes databases that do not require protection from SiteMinder. Resources that are not protected by SiteMinder are not authenticated as the default Domino user. Instead, the Domino server prompts users for their credentials (if anonymous access is disabled).

Modify the Domino Default User and the Domino Super User

To modify the DominoDefaultUser and DominoSuperUser parameters, do one of the following:

- Change it in the Agent Configuration Object, if configuring centrally

  You can modify the DominoDefaultUser and DominoSuperUser settings in the Agent Configuration Object. You can choose whether the values are encrypted or in plain text.

  Note: For more information, see the Policy Server documentation.

- Modify the parameters in the Agent configuration file using the encryptkey tool.

  In the Agent configuration file, the DominoDefaultUser and DominoSuperUser values must be encrypted. Consequently, you have to modify these values using the encryptkey tool.

  Important! Do not edit these settings directly in the Agent configuration file.

Use Encryptkey to Set the Domino Default or Super User

To set or change the value of DominoSuperUser or DominoDefaultUser in the Agent configuration file

1. Do one of the following:
   - UNIX: Navigate to the Domino Agent's bin directory. For example:

     /$HOME/ca/SiteMinder/Web Agent/bin
   - Windows: Open a command prompt window and navigate to the Domino Agent's Bin directory. For example:

     C:\Program Files\ca\SiteMinder Web Agent\Bin
2. Run the encryptkey tool, using the following arguments:
   - For DominoSuperUser:

     encryptkey -path *path_to_Agent_config_file*

     -dominoSuperUser *new_value*
   - For DominoDefaultUser:

     encryptkey -path *path_to_Agent_config_file*

     -dominoDefaultUser *new_value*

   For example:

   encryptkey -path "c:\program files\ca\SiteMinder Web Agent\Bin\Lotus Domino5\webagent.conf"

   -dominoSuperUser admin

   Note: The path to the Agent configuration file must contain the file name, such as, webagent.conf. Also, if any value in the path contains spaces, the entire path must be surrounded by quotation marks.

   Note: The encryptkey tool is not provided as a part of the SiteMinder Web Agent kit. However, the tool remains useful to Domino users who can manipulate it to generate encrypted DominoSuperUser settings for local configuration. You can contact Support to download a copy of this tool.

Map URLs for FCC Redirects

To accommodate URL conversion, you can configure the DominoMapUrlForRedirect parameter. It is enabled by default.

If this configuration parameter is set to YES, or does not exist, the Web Agent maps (normalizes) the URL from the Domino server representation to a URL-friendly name for Forms Credential Collector (FCC) redirects.

If this parameter is set to NO, the Web Agent does not map the URL and performs FCC redirects using the Domino server representation.

Coordinate SiteMinder and Domino Authentication

The following sections discuss coordination of SiteMinder and Domino authentication.

Use a SiteMinder Header for Authentication

The DominoUseHeaderForLogin and DominoLookUpHeaderForLogin parameters can be used to identify a Domino user for authentication.

DominoUseHeaderForLogin

Instructs the Domino Web Agent to pass the SiteMinder header value to the Domino Web Server. The Domino server uses the header data to identify a user in its user directory.

Set this parameter to a header name. For example, if you specify DominoUseHeaderForLogin="HTTP_SM_USER", the Web Agent passes the user's login name to the Domino server.

DominoLookUpHeaderForLogin

Instructs the Domino Web Agent to ask the Domino Web Server if the user requesting access to a resource is unique or ambiguous within the Domino user directory. This check is useful if a user named Jones tries accessing a resource and there are several users named Jones in the user directory. If this parameter is set to no, the Domino Web Agent does no checking with the Domino Web Server.

Default: Yes

Disable Domino Session Authentication

SiteMinder provides authentication and authorization functionality; therefore, the Domino session authentication feature is not needed. It should be disabled if the Web Agent is installed.

Under some conditions, having Domino session authentication enabled causes the user session to behave differently. This change in behavior does not affect security on a SiteMinder-enabled site. It reflects the intersection of SiteMinder and Domino session management rules.

Map URLs for FCC Redirects with a Domino Web Agent

If you are protecting Domino view (.nsf) resources with a forms authentication scheme, you need to map the URLs before they are redirected to the Forms Credential Collector.

To map the URLs before redirection

1. Set the DominoNormalizeUrls parameter to yes.
2. Set the DominoMapUrlForRedirect parameter to yes.

   Domino URLS will be mapped before redirection to the Forms Credential Collector.

Disable URL Normailization

The process of URL normalization modifies URLs from a Domino representation to a URL format used by a typical web browser. The Domino Web Agent relies on the Domino web server APIs to normalize a Domino URL.

During the normalization process, the Domino Server APIs periodically return a URL with a carriage return (0x0D in hex) and/or a line feed character (0x0A in hex) added to the normalized URL. The addition of these characters appears to be related to specific Notes database (.nsf) files and access patterns within these files.

The following example shows a normalized URL with an added carriage return:

- URL:
  http://server.ca.com:80/agentrunner.nsf/be68f4545348400461332?OpenView
- URL is mapped to:
  http://server.ca.com:80/agentrunner.nsf/AgentContext?OpenView
- URL is normalized to:
  http://xxxxx.ca.com:port/agentrunner.nsf/0x0d/AgentContext?OpenView

If necessary, you can ensure that URLs with Domino resource IDs are not normalized with the following parameter:

DominoNormalizeUrls

Specifies if the SiteMinder Web Agent converts Domino URLs to a URL-friendly name before redirecting them to a Forms Credential Collector.

The MapUrlsForRedirect parameter must also be set to yes for the Domino URLs to be converted.

If the DominoNormalizeUrls parameter is set to no, URLs will *not* be normalized, even if the MapUrlsForRedirect parameter is set to yes.

Important! If you set the DominoNormalizeUrls parameter to no, you cannot protect individual documents within a Notes database; you can only protect the entire database or subdirectories of the Domino Web server.

Default: Yes

To turn off normalization and ensure that URLs are not altered, set the DominoNormalizeUrls parameter to no.

Control Access to Lotus Notes Documents

The Web Agent offers a finer level of granularity for protecting Lotus Notes documents on Domino. The folloiwng parameter controls this protection:

DominoLegacyDocumentSupport

Specifies how a Web Agent handles user requests for protected Lotus Notes documents in a Domino environment. Setting this parameter to yes grants users ReadForm permission only for the requested document.

Default: No

Use the DominoLegacyDocumentSupport parameter to configure the Web Agent to process user-requested actions when accessing Notes documents. This offers a finer granularity of protection on Domino.

Notes documents do not have names. They are saved to the database with a reference to the form used to create them. When a user requests a Notes document, the Domino Web Agent finds the form for that document by converting the request into a URL. This URL includes the original Domino action. If no form is found, then nothing is used.

For example:

"http://server.domain.com/db.nsf?OpenDocument"

in the URL To ensure that the Web Agent performs the user-requested Domino action on the document that is specified in the URL, such as ?OpenDocument or ?EditDocument, set the DominoLegacyDocumentSupport parameter to no.

For example, if the URL request is:

http://www.dominoserver.com/names.nsf/9348730948938987785784395880982039857983 49?EditDocument

The Domino Agent converts the preceding URL to:

http://www.dominoserver.com/names.nsf/Person?EditDocument

where Person is the name of the form used to create the document identified by the NotesID in the original URL.

To force the Domino Web Agent revert back to its pre-4.6 operation for accessing Notes documents, which means that only the action ?ReadForm is permitted, set this parameter to yes. With the legacy document support enabled, the Domino Agent would convert the URL in the previous example to:

http://www.dominoserver.com/names.nsf/Person?ReadForm

Enable a Domino Agent to Collect Credentials for Authentication

A credential collector is an application within the Web Agent, which gathers user credentials for forms, SSL, and Windows authentication schemes, and for single sign-on across multiple cookie domains. The credentials gathered by the credential collector are based on the type of authentication scheme configured for a particular group of protected resources.

For a Domino Web Agent to act as a credential collector, you have to configure various MIME types, represented as file extensions in the Agent configuration file.

Credential collectors are generally auto-authorized, that is, when you add a file extension to these parameters, they are, by default, included in the IgnoreExt parameter. Domino Server cannot correctly process URLs that include files with these extensions, so the Domino Agent has to ignore these files.

Note: For more information, see the Policy Server documentation.

More Information

Use Credential Collectors for Authentication and Single Sign-On (see page 231)

Specify User Directories for Domino

The Domino Directory is integrated with every Domino server. You can enable LDAP service for the Domino server so that Policy Server can use the Domino Directory to authenticate and authorize users. If you enable Domino's LDAP service, you do not need to configure a separate user directory for authentication.

To enable LDAP service, see your Domino Server documentation.

To use Domino's LDAP directory as a user store, use the Domino Application Server, minimum version 5.02.

Note: For more information, see the Policy Server documentation.

More information:

Contact CA (see page iii)

Configure Policies for Domino

The Domino server can represent the same Notes object in different ways. An object can be identified using the name, ReplicaID, UniversalID, and alias.

For the Domino Web Agent to communicate effectively with the Domino server, the Domino Agent processes access requests to Notes resources using only the object name. This enables the SiteMinder policy store to understand the entry.

Expressed as a URL, the access method to any resource would be:

http://host/database.nsf/resource_name?Open

Create Rules for Domino Server Resources

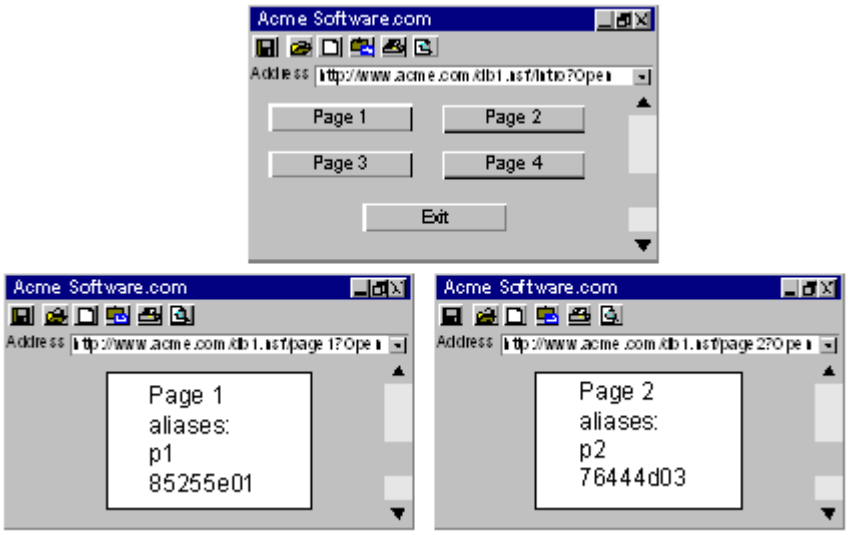
Actions for the Notes database resources should be considered when you create rules. Any resource not specified with an action will default to the action ?Open. The rules that are included in a SiteMinder policy must account for the default action, ?Open, and equivalent actions for ?Open, such as ?OpenDatabase, ?OpenView, ?OpenDocument, ?OpenFrameset.

The Domino Web Agent enables a policy administrator to create one rule for many aliases that point to the same resource. You only need one rule because the Domino Agent converts Domino's multiple representations of a resource into one URL. This function of the Domino Agent is important to consider when creating rules for SiteMinder policies.

You create realms and rules using the Administrative UI.

Note: For more information, see the Policy Server documentation.

In the following illustration, the URL is a link to Acme's Domino server, with a Notes database called db1.nsf. This database contains two files: page1 and page2.



Example 1: Protecting one document and all its aliases.

For access to page1 and all its aliases, you create only one rule for the realm db1.nsf. The Domino Agent is able to interpret all the different naming conventions and convert them to a one standard URL format.

For your realms and rules, do the following:

- When creating a realm you would specify a resource filter for the database where page1 resides. For example, to protect all files in the database you would configure the following:

  Resource filter: /db1.nsf/

  To protect not only page1 but all its aliases, you would configure the following:

  Resource filter: /db1.nsf/page1

- To create a rule that protects any action on page1, enter an asterisk (*) in the Resource field of the Rule Properties dialog box. For example:

  Resource: *

  This * wildcard indicates that any action, such as ?Open, ?EditDocument can be performed on page1 by the users that are bound to the policy.

Example 2: Protecting different documents in the same database.

To protect page2 in the db1.nsf database in addition to page1, you need to create a second rule.

Resource Filter: /db1.nsf/page2

Resource: *

Example 3: Protecting different actions on a single resource

To protect individual actions on a resource, for example, if you wanted only some users to perform the action ?EditDocument and all users to perform the action ?ReadForm, each action would require its own rule for each resource, as follows:

- Rule 1

  Resource Filter: /db1.nsf/page1

  Resource: ?OpenView

- Rule 2

  Resource Filter: /db1.nsf/page1

  Resource: ?EditDocument

You could also use one rule as follows:

Resource Filter: /db1.nsf/page

Resource: ?Open*

Note: In the Resource field, there is no forward slash (/) before ?Open.

Even if there are aliases for this resource, the one rule would protect the original page and all its aliases.

Instead of creating several rules for different actions, you could specify a single rule and use wildcards to cover all actions, for example:

Resource filter: /db1.nsf/page

Resource: ?Open*

With the rule, you are then protecting the resource:

http://www.acme.com/db1.nsf/page*?Open*

Note: If you want a rule to be literal, write a regular expression.

Configure Full Logoff Support for Domino Agents

Full logoff uses a custom logoff page that you create along with the following parameter:

LogOffUri

Enables full log off and specifies the location of a custom web page on your web server that appears to users after they are successfully logged off. You must configure this page so that it cannot be stored in a browser cache. Otherwise, a browser may display a logoff page from its cache without logging the user off. This may give an unauthorized user an opportunity to assume control of a session.

Note: When the CookiePath parameter is set, the value of the LogOffUri parameter must point to the same cookie path. For example, if the value of your CookiePath parameter is set to example.com, then your LogOffUri must point to example.com/logoff.html

Default: No default

Limits: Do *not* use a fully qualified URL.You must use a relative URI.

Example: /Web pages/logoff.html

To configure full logoff

1. Create a custom HTTP application to log the user off, for example, add an Exit or Sign Off button that redirects the user to a URL you specify.
2. To make sure that an HTML logoff page is loaded from the web server and not from the browser's cache, set up the logoff page so it cannot be cached in the browser. For example, for HTML pages, you can add the following meta tags to the page:

   ```
   <META HTTP-EQUIV="Pragma" CONTENT="no-cache">
   ```

   ```
   <META HTTP-EQUIV="Expires" CONTENT="-1">
   ```

   Important! Meta tags may not always work with an Internet Explorer browser. In this case, use a cache-control HTTP header.
3. Configure the LogOffURI parameter with the following steps:
   a. Delete the pound sign (#), if necessary.
   b. Enter the URI of the custom HTTP file that will log the user off. Do not enter a fully qualified URL.

      Full logoff is configured.

More Information

How Full Logoff Works (see page 106)

Considerations for Creating Policies on Domino Servers

Consider the following when creating SiteMinder policies for the Domino server:

- A user can open a form with a parent document to view default values for the form. The parent document is the original form used to create the document. To prevent a user from viewing default values on a form they should not have access to, set the SkipDominoAuth parameter to no, which prevents the user from having Super User privileges.
- If you replicate databases on the same machine, create a duplicate set of rules to protect each database.
- If the Domino Agent cannot associate an alias for a Notes document with a form, then each document requires its own rule for protection.
- The Domino server uses special identifiers in URL commands for certain database documents, for example, $DefaultView, $DefaultForm, $DefaultNav, and $SearchForm. The Domino Web Agent converts these identifiers to a standard URL to access the document. In the case of $defaultNav, the Domino Agent performs an action of ?OpenDatabase. You do not need to create additional rules for these types of identifiers.
- Resources in the Notes database are protected by their alias. If the alias does not exist, it is protected by the resource name or comment.
- Notes allows multiple objects of different types to have the same name and alias. If you create a rule that uses a wildcard with the ?Open action, such as, ?Open*, be aware that this rule protects different types of resources that share the same alias or name.
- Documents are protected by the form with which they were created. The action used with the form is ?ReadForm.
- The Domino Agent will not ignore files with the .nsf extension. Do not add this extension to the IgnoreExt parameter.

Use a Domino Agent with a WebSphere Application Server

A Domino web server acts as the front end to a WebSphere Application Server by providing a filter plug-in that intercepts requests before forwarding them to the WebSphere server.

Chapter 22: Advanced Authentication Scheme Configuration

This section contains the following topics:

Protect IIS 6.0 Web Server Resources with Passport Authentication

For IIS 6.0 web server resources protected by Passport authentication, you must enable the Passport application on the IIS 6.0 web server itself in addition to performing the SiteMinder configuration steps for Passport authentication.

To enable the Passport authentication

1. Open the IIS Manager.
2. From the appropriate website, right-click the directory or file for which the Passport application should be enabled and select Properties.

   The Properties dialog box opens.
3. Do one of the following tasks:
   - If you right-clicked a directory in Step 2, click the Directory Security tab.
   - If you right-clicked a file in Step 2, click the File Security tab.

   The appropriate tab opens.
4. Click Edit.

   The Authentication Methods dialog appears.
5. Select the .NET Passport authentication check box.
6. Click OK.

   The Authentication Methods dialog closes.
7. Click OK.

   The dialog closes and Passport Authentication is enabled for the item you chose.

Delete Certificates from Stronghold (Apache Agent Only)

Stronghold web servers write client certificates to a local, temporary file, which the Web Agent uses for certificate-based authentication. The Stronghold server uses this file to make information in the client certificate available for authentication. As users visit a website, these certificate files increase, taking up space on your server. You can configure the Web Agent to delete a certificate file after the Agent has finished using it.

To delete certificate files, set the DeleteCerts parameter to yes.

Accommodate Legacy URL Encoding

The legacy URL encoding used by CA uses dollar sign ($) characters. If the dollar signs cause problems, you can make the Web Agent use hyphen (-) characters instead of dollar signs with the following parameter:

LegacyEncoding

Forces the Web Agent to replace any dollar sign ($) characters in legacy URLs with a hyphen (-). This also ensures backwards comparability with MSR, Password Services, and DMS. When this parameter is set to no, a Web Agent converts the string SM to -SM-. When this parameter is set to yes, the Web Agent does *not* convert the dollar sign ($) character.

Default: (Framework Agents) No

Default: (Traditional Agents) Yes

To encode legacy URLs using hyphens instead of dollar signs, set the value of the LegacyEncoding parameter to no.

Chapter 23: Single Sign-On Security Zones

This section contains the following topics:

Security Zone Definitions

The following terms apply to single sign-on (SSO) security zones:

CAC (Centralized Agent Configuration)

Identifies a mechanism by which a Web Agent borrows its configuration properties from a Web Agent configuration object defined in the policy store.

Cookie Provider

Identifies a mechanism by which single sign-on is implemented in Web Agents across multiple domains. One of the domains is designated as the master domain, and the Web Agents from the other domains are re-directed to a Web Agent in the master domain to provide them with the cookies in that domain.

SSO (Single Sign-On)

Identifies a mechanism by which a user authenticated once will not be rechallenged for credentials.

SSO Zone

Identifies a sub-set of SSO, defined by an arbitrary identifier (zone name) used to segment application SSO within a single cookie domain. All applications in the same SSO zone allow SSO amongst themselves. SSO to and from other SSO zones may or may not be allowed as defined by zone trust relationships.

Trusted SSO Zone

Identifies a foreign zone trusted by a local zone for SSO.

Security Zones Overview

Users have the ability to define single sign-on security zones within the same cookie domain, representing a single zone, or across multiple cookie domains, representing different zones. As a result, users have single sign-on within the same zone, but may be re-challenged when entering a different zone, depending on the trust relationship defined between the zones. Zones included in a trusted relationship will not re-challenge a user that has a valid session in any zone in the group.

Single sign-on security zones are implemented entirely by SiteMinder Web Agents. Each zone must reside on a separate Web Agent instance. Multiple zones cannot be created on the same Agent instance.

A security zone is identified by cookies generated by the Web Agent. By default, the Web Agent generates two cookies, a session cookie named SMSESSION and an identity cookie named SMIDENTITY. When you configure security zones, the Web Agent can generate session cookies and identity cookies with unique names so that the zone affiliation is reflected in the cookie names.

Security Zones Benefits

The SSO Security Zones feature is intended for use in situations where SiteMinder administrators wish to segment their single sign-on environments within the same cookie domain. For example, consider the CA.COM domain. Under standard SiteMinder SSO functionality, all SiteMinder protected applications in CA.COM would use the cookie SMSESSION to manage single sign-on. Consider the following scenario in which SSO Security Zones do not exist:

1. A user accesses an application (APP1). The user is challenged by SiteMinder, logs in to SiteMinder, and creates an SMSESSION cookie.
2. The user accesses a second application (APP2) and is once again challenged by SiteMinder. (Rules prevent SSO from occurring because the user does not have access to APP2 using the APP1 user credentials.) The user logs in and creates a new SMSESSION cookie overwriting the old one with the new logged in session for APP2.
3. The user now returns to APP1 and is challenged yet again, since the user lost the original APP1 session and the APP2 session might not be accepted for APP1. Therefore, SSO does not occur between APP1 and APP2, causing a very frustrating situation.

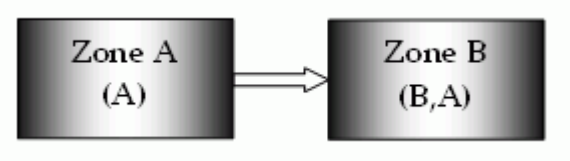
With SSO Security Zones, APP1 can be placed in zone Z1 and APP2 can be placed in zone Z2. Now logging into APP1 creates a Z1SESSION cookie and access to APP2 results in a Z2SESSION cookie. With different names, the cookies no longer overwrite each other so there is only one login per application now, not one for each time the user moves between the applications as in the example above.

Prior to the SSO Security Zones feature, the only way to perform the same grouping of SSO for applications was to create different network domains and therefore different cookie domains (CA1.COM, CA2.COM, and so on), and use various multi-cookie domain configurations with cookie providers. This is not desirable in most enterprises, since using multiple network domains has certain IT maintenance and support consequences.

Security Zone Basic Use Case

Single sign-on can, on a controlled basis, be broken into several security zones that have configurable trust relationships. For example, consider Zone A and Zone B:

- Zone A has only one trusted zone, its own Zone A.
- Zone B has two trusted zones, its own Zone B as well as Zone A.



The trust relationship in the above illustration is indicated by the arrow, meaning that the user sessions established in Zone A can be used for single sign-on in Zone B.

In this example, Zone A might be an administrator-only zone, while Zone B might be a common access zone. An administrator authenticated in Zone A gains access to Zone B without being rechallenged. However, a user authenticated in Zone B is re-challenged when trying to access Zone A.

User sessions in different zones are independent of each other. Suppose a user authenticates in Zone B first, and then authenticates again in Zone B. Two different sessions are created. In fact, the user may have different identities in both sessions. When the user returns to Zone A, the session established in that zone is used.

Consider what would happen if a user is validated using single sign-on in a zone where that user does not yet have a session. If the user authenticates in Zone A and then visits Zone B for the first time, then a user session is created in Zone B, based on the session information in Zone A, possibly updated by the Policy Server. Note that the user session in Zone A is not updated until the user returns to Zone A.

User Sessions Across Security Zones

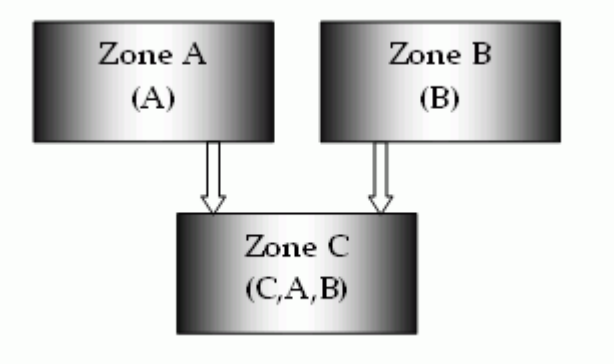
Single sign-on security zones do not need to all belong to the same domain. In fact, zones can be spanned over multiple domains. However, Web Agents only search for trusted zone cookies in their local cookie domain. If no suitable cookies are found, Web Agents continue to redirect to cookie providers only for their own zone.

Trusted Zone Order

A single sign-on security zone is defined by a pair of parameters:

- the security zone name
- an ordered list of trusted zones

The order in which the trusted zones are listed is important. Consider the following example.



In this illustration, Zone C trusts both Zone A and Zone B. Neither Zone A nor Zone B trusts any other zone, but all zones trust themselves.

When a user makes a request in Zone C, the Web Agent looks for a session or identity cookie in the trusted zones, in the order in which the zones are listed. In this example, Zone C has a list of trusted zones that include C, A, and B.

The following is an order of events that might occur:

1. The Web Agent first checks to see if the user has a session in Zone C.
2. If no session is found, the Web Agent checks to see if the user has a session in Zone A.
3. If no session is found, the Web Agent checks to see if the user has a session in Zone B.
4. The session specification from each cookie that is found is used to process authentication requests until a successful login occurs.
5. After a successful authentication, the Web Agent proceeds to authorization.
6. If no cookies are found or no cookies pass authentication, the agent challenges the user for credentials as usual.

Note that the user experience may depend on the order in which the zones are accessed. In this example, if the user accesses Zone B first followed by Zone C, the user's identity in Zone B is also used in Zone C. If the user accesses Zone A first followed by Zone B and Zone C, the user's identity in Zone A is used, despite the fact that the user was re-challenged in Zone B before going to Zone C.

This will also be the case when sessions with different max and idle session timeouts begin to expire. In the current example, a user with valid cookies in Zone A and Zone C will first get access with the Zone C cookie. If the Zone C cookie expires, the Zone A cookie will be used if it has not expired. Therefore, the user's identity could change from a Zone C identity to a Zone A identity without a credential challenge occurring.

Two or more Web Agents may have different lists of trusted zones but still use a common trusted zone name. In this case, the agents implicitly trust each other but will not trust the same foreign zones. This functionality enables applications to be segmented for single sign-on. A Web Agent supports only a single sign-on zone name. All session, identity, and state cookies generated by that agent use the same single sign-on zone name. Therefore, if two applications do not share the same single sign-on trust requirements, they must be hosted on separate web servers each with its own Web Agent and list of trusted zones.

Note: Foreign zones refer to zones other than the one supported by a given Web Agent. For example, if an agent is configured with SSOZoneName="Z1", then any other zone would be foreign to it. This includes the default zone "SM".

The Default Single Sign-On Zone and Trusted Zone List

Web Agents that do not specify a security zone name (such as all pre-SiteMinder 6.x QMR 5 Web Agents) are considered to belong to the default zone. For backwards compatibility, the default zone is implicitly assumed to have a zone name of SM. This allows SiteMinder r12.0 SP2 Web Agents to support SMSESSION and SMIDENTITY by default with no configuration changes.

Web Agents that do not specify a list of trusted zones trust only their own single sign-on zone (either a specified zone name or default zone if no zone name has been specified).

A Web Agent can be configured to trust other zones in addition to the default zone. It can also use a specified zone name and list no other trusted zone. Agents always trust their own zone first, regardless of whether or not additional trusted zones are specified. In order for a Web Agent using a non-default zone to trust the default zone as well, it must list "SM" in its trusted zone list.

Request Processing with Multiple User Sessions

Web Agents look up user sessions in the order of trusted zones. If a valid user session is found, the Web Agent uses the session information to process the user request. If no valid session is found, the Web Agent fails over to the next valid user session in the order of trust.

Responses from failed validations are ignored if there is another session to check. Otherwise, they are processed as normal. This means that if the Web Agent finds three trusted sessions to process and the first two fail to validate, only the responses from validating the third and final session are processed.

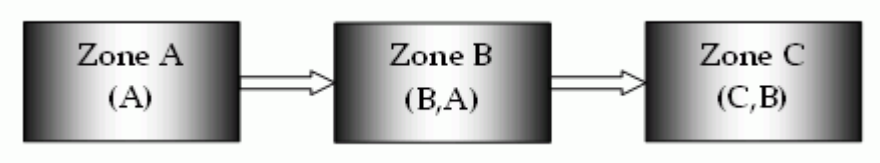
In the case of a successful validation, the Web Agent stops processing sessions and immediately begins processing the responses from the successful validation. If the agent has three sessions to validate and the first session validates successfully, the remaining two are ignored and the agent moves on to process the responses for the first successful validation.

More Information

Trusted Zone Order (see page 285)

Transitive Relationships Across Zones

The trust relationship between single sign-on zones is not fully transitive. If Zone A is trusted by Zone B, and Zone B is trusted by Zone C, Zone A may not necessarily be trusted by Zone C, as illustrated in the following diagram.



Other Cookies Affected by Single Sign-On Zones

SiteMinder uses state cookies to manage the various events surrounding authentication and authorization. All of these cookies by default begin with the default single sign-on security zone prefix SM. If a new single sign-on zone name is specified, then these cookies are also named to reflect the specified non-default zone name. Below is a list of cookies that are affected by defining a new single sign-on zone:

- SMCHALLENGE
- SMDATA
- SMIDENTITY
- SMONDENIEDREDIR
- SMSESSION
- SMTRYNO

If a zone name of Z1 is specified, for example, the Web Agent begins creating Z1CHALLENGE=YES cookies for Basic authentication. This allows administrators to create islands of SiteMinder application single sign-on in a single cookie domain (for example, ca.com) where agents will not interfere with each other. The single sign-on trusted zone list then allows single sign-on to occur between these isolated single sign-on zones in a predictable fashion.

Single Sign-On Zones and Authorization

With single sign-on zones, authorization proceeds normally after a successful authentication without change. Once the validation process identifies a valid session, the session is used for the remainder of request processing and any other sessions identified in the request are ignored. If authorization fails, the user is challenged regardless of whether or not other sessions are available that might authorize successfully.

The first trusted session that passes validation is the session passed to authorization. If this session fails authorization, the user is challenged for credentials.

Configure Security Zones

Two single sign-on parameters have been added to the Web Agent configuration objects in the policy store. These settings may also be used in local configuration files and are added to the sample local configuration files laid down during installation.

Note: All Web Agents configured through the same agent configuration object belong to the same single sign-on zone.

SSOZoneName

Specifies the (case-sensitive) name of the single sign-on security zone a Web Agent supports. The value of this parameter is prepended to the name of the cookie a Web Agent creates. This helps you associate cookies with their respective cookie domains. When this parameter is not empty, SiteMinder generates cookies using the following convention:

ZonenameCookiename.

Default: Empty (uses SM as a zone name, which gives the cookies the following default names):

- SMSESSION
- SMIDENTITY
- SMDATA
- SMTRYNO
- SMCHALLENGE
- SMONDENIEDREDIR

Limits: Single-valued

Example: Setting the value to Z1 creates the following cookies:

- Z1SESSION
- Z1IDENTITY
- Z1DATA
- Z1TRYNO
- Z1CHALLENGE
- Z1ONDENIEDREDIR

SSOTrustedZone

Defines an ordered (case-sensitive) list of trusted SSOZoneNames of trust for a single sign-on security zone. Use SM to add the default zone if necessary. Agents always trust their own SSOZoneName above all other trusted single sign-on zones.

Default: Empty (SM or the SSOZoneName if provided)

Limits: Multi-valued

Specify the Single Sign-on Zone for the Agent

SSOZoneName

Specifies the (case-sensitive) name of the single sign-on security zone a Web Agent supports. The value of this parameter is prepended to the name of the cookie a Web Agent creates. This helps you associate cookies with their respective cookie domains. When this parameter is not empty, SiteMinder generates cookies using the following convention:

ZonenameCookiename.

Default: Empty (uses SM as a zone name, which gives the cookies the following default names):

- SMSESSION
- SMIDENTITY
- SMDATA
- SMTRYNO
- SMCHALLENGE
- SMONDENIEDREDIR

Limits: Single-valued

Example: Setting the value to Z1 creates the following cookies:

- Z1SESSION
- Z1IDENTITY
- Z1DATA
- Z1TRYNO
- Z1CHALLENGE
- Z1ONDENIEDREDIR

Use the SSOZoneName parameter to enter the name of the single sign-on zone a Web Agent is to support. This parameter is case sensitive. If not specified, it defaults to SM. If the value of the SSOZoneName parameter is non-empty, the Web Agent generates cookies with the naming convention:

zone_namecookie_name

where *zone_name* is the parameter value and *cookie_name* is the general name of the cookie being created.

Cookies affected by this convention include:

- SESSION
- IDENTITY
- DATA

- TRYNO
- CHALLENGE
- ONDENIEDREDIR

If the user is validated in a single sign-on zone in which that user has not yet established a session, the session specification returned by the Policy Server is used to create a new session cookie for that zone.

When a new cookie is created, its zone parameter is set to the zone name, in order to prevent the user from swapping cookies from different zones by simply renaming them. The cookie validation engine verifies if the zone name matches the prefix used in the cookie's name. This applies only to SESSION and IDENTITY cookies.

To specify the name of the single sign on zone you want the Web Agent to support, add the name of the zone to the SSOZoneName parameter.

The Order of Trust and Failover

Use the SSOTrustedZone parameter to specify the single sign-on zone's order of trust. When processing a request, the Web Agent looks for a SESSION or IDENTITY cookie for each zone in the order they appear in the list.

Any cookies found are validated as usual (decrypted, and tested for a valid host name, single sign-on zone name, and timeouts), then stored in an ordered list of trusted sessions if valid. Prior to authentication, the user's active session (and therefore user identity) are considered the first session in the ordered list of valid sessions.

During authentication, the Web Agent will call validate using the first session in the list. If the validation succeeds, the agent moves on and establishes user identity and affirms the active accordingly. If validation fails, the next session is used in a new validation call, and so forth until validation succeeds or the agent runs out of sessions. If no session validates, the agent challenges the user as usual.

Once validated, the agent ignores all other sessions and instead sticks only to the session that validated for the remainder of request processing. This means that should authorization fail, the user is challenged immediately. Any other existing sessions in the request are not used.

Appendix A: Troubleshooting

This section contains the following topics:

Agent Won't Start Because LLAWP is Already Running or Log Messages not Written to the Correct Log Files

Valid on UNIX

Symptom:

I'm having one or more of the following problems:

- My Web Agent won't start because the LLAWP process is already running.
- My Web Agent starts, however the log messages are being written to the log files of a second agent instance.

Solution:

This problem may occur when multiple disks on the same computer use the same mount point. The Web Agent uses the inode of a directory to allocate system resources, and if the inodes are the same, resource collisions and errors result. To fix this problem, use the following process:

1. Create a new subdirectory on your web server (this creates a unique inode).
2. Change the path shown in the ServerPath parameter of the Web Agent so it points to the new subdirectory.

More information:

Set the ServerPath Parameter for UNIX Systems (see page 55)

Web Server Does Not Prompt for Username or Password

Symptom:

The web server does not prompt me for a username and password.

Solution:

Do the following:

- Check if the resource is defined correctly in the Policy Server.
- Make sure the Web Agent is enabled.
- Make sure that the realm and rule are defined properly.

Web Server Authentication Fails

Symptom:

The web server always prompts me for a username and password, but authentication fails.

Solution:

Do the following:

- Check if the resource is defined correctly in the Policy Server.
- Make sure that a user has been bound to a policy.
- Check that a rule is defined properly for the policy.

Server Error 500 Appears Instead of Custom Error Page

Valid on Windows

Symptom:

When a server error occurs, I see a generic 500 error in my browser instead of my custom error message.

Solution:

Disable the Show Friendly HTTP Error Messages option in Microsoft Internet Explorer.

Note: For more information, see your Microsoft documentation, or go to http://support.microsoft.com/

More information:

Custom Error Handling For Applications (see page 139)

Configured Attributes Are Not Reaching Web Application

Symptom:

The attributes I set up are not reaching my web application.

Solution:

Check if the resource is defined correctly in the Policy Server, as follows:

- For CGI scripts, add the **HTTP_** prefix to the attribute names when querying them from the application, for example, The attribute CAN_READ should be defined in the Policy Server as HTTP_CAN_READ.
- For Java servlets, make sure that HTTP_ is *not* added to the front of the attribute name.
- For ASP pages, parse the ALL_HTTP variables.

Using the SiteMinder Test Tool, check whether or not your attributes are displayed in the Attributes field of the Server Response group box.

Agent Is Sending Authorization Requests Configured to Ignore to Policy Server

Symptom:

The Agent is sending authorization requests to the Policy Server for extensions the Agent is configured to ignore.

Solution:

Check the values for IgnoreExt. The string must be comma-separated and without spaces. If there is more than one section in the URL that contains periods, the Web Agent will always send an authorization request.

Browser Is Not Submitting Cookie

Symptom:

My browser is not submitting the cookie to my other web server after I authenticated at the first server.

Solution:

Check that the CookieDomain field for single sign-on contains at least two periods and that it is a subset of the server name, as in the following example:

.mycompany.com cookie domain works for all machines that end in .mycompany.com. Although it is possible to access a local web server without the full domain name, this method cannot generate the submission of a cookie by your browser. Therefore, the URL http://www/index.html gets the web page, but cannot match the cookie domain, so a cookie is not submitted (you may be asked to reauthenticate).

If you are using a country code, for example, www.company.abc.uk, make sure there are three periods in the URL. Also, be aware of the following:

- Check that you specified the fully qualified domain name in the browser.
- If you set CookieDomain to none, the Web Agent creates cookies only for the server that issued them.
- If you leave CookieDomain blank, the Web Agent gets the cookie domain from the HTTP_HOST header based on the value of the CookieDomainScope parameter.
- When the CookieDomainScope parameter is set to 0, the Agent chases the most specific cookie domain for the host without making a server-only cookie. This means that the host myserver.netegrity.com yields a domain of netegrity.com, and myserver.metals.ge.com yields a cookie domain of .metals.ge.com. If CookieDomainScope is set to 2, the cookie domain would be .netegrity.com and .ge.com respectively.

Solaris/Sun Java System Web Agent Not Loading or Web Server Not Starting

Symptom:

The Solaris/Sun Java System Web Agent is not loading or the web server is not starting, and the lines are properly entered in the obj.conf file.

Solution:

Check the error log for the server in the following file:

/https-*server_name*/logs/errors

Receive WriteLine Failed Error

Symptom:

I reinstalled the Windows/Sun Java System Web Agent, but I get a WriteLine Failed error stating that it "Could not write hostname into the Webagent.conf file, error -1".

Solution:

Make sure you removed the WebAgent.conf file from the web server sub-directory after uninstalling the Web Agent.

Solaris/Sun Java System Web Agent Not Communicating with Policy Server

Symptom:

The Solaris/Sun Java System Web Agent is loading, but it is not communicating with the Policy Server.

Solution:

Do the following:

- Make sure the Agent configuration file includes the EnableWebAgent parameter and it is set to yes. The default setting is no.
- Check that the Agent has TCP connectivity to the Policy Server. If there is a firewall between the Web Agent and the Policy Server, be sure that TCP ports 44441, 44442, and 44443 are not blocked by the firewall for two-way traffic.

Apache Web Server Will Not Start/Restart when Web Agent is Enabled

Symptom:

The Apache web server will not start or restart when the Web Agent is enabled.

Solution:

If the web server will not start, do the following:

- Check the Apache error log located in *apache_server_home*/logs/error_log and look for SiteMinder errors. The Apache Web Agent may be trying to allocate more shared memory or semaphores than your kernel is set to allow.
- Tune your Solaris operating system for the Apache Web Agent.

  Note: For more information, see the SiteMinder Web Agent Installation documentation.
- If you start your Apache web server from multiple user accounts, you may have orphaned semaphores on your system. Reboot or use the ipcrm -s command to remove the orphaned semaphores. We recommend always starting the web server from the same user account.

If the web server will not restart, do not use the restart command. Use the stop and start commands to restart the server.

Apache Reverse Proxy Server Shows a 500 Error When the URL Contains the % Character

Symptom:

I am using an Apache web server as a reverse proxy server (the ProxyPass directive is enabled in the httpd.conf file), but requests for URLs that contain the percent (%) character return an HTTP 500 error.

Solution:

Remove the %25 from the list of characters used as the value of the BadURLChars parameter in your Agent Configuration Object or local configuration file.

Sun Java System Web Agent on Solaris Not Loading

Symptom:

The Sun Java System Web Agent on Solaris is not loading or not communicating with the Policy Server.

Solution:

Do the following:

- Make sure the Init statement for WebAgent.so.1 is in the obj.conf file and in the section with the rest of the Init statements. This line must contain a valid path to the WebAgent.so.1 file and contain the two functions SiteMinder Agent and SmRequireAuth in quotes, separated by a comma only, no spaces, as in the following example:

  ```
  Init fn="load-modules" shlib="/usr/iPlanet/servers/https-myserver/config/WebAgent.so.1"
  funcs="SiteMinderAgent,SmRequireAuth"
  ```

- Ensure the Web Agent name matches what is specified in the Agent Configuration Object at the Policy Server.
- Check that the EnableWebAgent parameter is set to yes.
- Make sure the AuthTrans statement exists in the default object and references the SiteMinderAgent function. If there are other AuthTrans lines in the obj.conf file and the Web Agent is not working, try commenting them out and using only the WebAgent AuthTrans function, as in the following example:

  ```
  AuthTrans fn="SiteMinderAgent"
  ```

- Make sure the PathCheck statement exists in the default object and follows the rest of the PathCheck lines in the object. It should reference the function SmRequireAuth, as in the following example:

  ```
  PathCheck fn="SmRequireAuth"
  ```

iPlanet WebServer Shows Blank Page when Using Basic over SSL

Symptom:

iPlanet WebServer shows a blank page when using Basic over SSL. The way Microsoft Internet Explorer (MSIE) handles SSL version 3 (SSLv3) and Transport

Transport Layer Security (TLS) keep-alive connections cause interoperability problems with non-Microsoft web servers such as the iPlanet web server. When accessing a web server over SSL (https://) connections, Internet Explorer may inappropriately display error messages or blank pages.

Solution:

iPlanet Web Server 6.0 SP2 introduces new functionality to work around this problem. The following two remedies are possible:

- Add the following line immediately below the <Object name="default"> line in the server's obj.conf files:

  ```
  AuthTrans fn="match-browser" browser="*MSIE*" ssl-unclean-shutdown="true"
  ```

  This line instructs the server to not send a close_notify alert when it closes SSLv3 connections from MSIE browsers. The close_notify packet is a required component of the SSLv3 and TLS specifications, but it is misinterpreted by MSIE.

  Note: The close_notify packet is used in SSLv3 and TLS connections to inform the other party in the transaction that the connection is being closed. Instructing iPlanet WebServer to not send the close_notify packet may make MSIE vulnerable to a truncation attack.

- Add the following line immediately below the <Object name="default"> line in the server's obj.conf files:

  ```
  AuthTrans fn="match-browser" browser="*MSIE*" keep-alive="disabled"
  ```

  This line instructs the server to disable keep-alive connections for Internet Explorer browsers. Disabling keep-alive connections may decrease your server's performance.

Sun Java System Web Server Restarts After a Forms-based Authentication Request

Valid on Solaris

Symptom:

When SiteMinder makes an authentication request using forms, the web server dumps its core and restarts.

Solution:

Increase the value of the StackSize parameter in the magnus.conf file.

Example: If the StackSize was set to 131072 try increasing the setting to 262144.

Authorization Requests Fail

Symptom:

The authorization requests made by the Web Agent are failing.

Solution:

Ensure that the key store is available.

Note: For more information, see the Policy Server documentation.

Web Agent Processes Client Requests Twice

Valid on SunOne Web Server 6.1 with SiteMinder Web Agent r6.x QMR5 or higher

Symptom:

After upgrading the Web Agent on my SunOne 6.1 web server to SiteMinder r6.x QMR5 (or higher), the Web Agent exhibits one or more of the following behaviors:

- Client requests pass through the Web Agent twice. The first request uses HTTP 1.0 or 1.1, and the second request uses HTTP 0.9.
- The Web Agent unexpectedly sets valid request variables (rq->vars) in the request as HTTP headers.
- Custom headers now use the the NAME=*Value* format, instead of the HTTP_NAME=*Value* format used previously, and my application that uses headers in the latter format no longer works.

Solution:

Set the value of the DisableDirectoryList parameter to yes.

More information:

Restrict Directory Browsing on a Sun Java System Server (see page 63)

Response Text not Displayed when OnAuthAccept rule and OnAuthAcceptText Response used with FCC

Symptom:

When the OnAuthAccept rule is paired with an OnAcceptText response and the user is authenticated using a SiteMinder FCC, the text in the response is not displayed.

Solution:

This behavior occurs when the FCC Compatibility Mode (FCCCompatMode) parameter is set to the (default) value of no. When the value is no, user authentication takes place at the Forms Credential Collector (FCC), where the response is triggered, but the text is lost when the FCC redirects the user back to the requested resource. The Web Agent validates the user's session, but since this is *not* an authentication event, the OnAcceptText response is *not* triggered.

To change this behavior, set the value of the FCCCompatMode parameter to yes.

Appendix B: Web Agent Configuration Parameters

This section contains the following topics:

Agent Configuration Parameters

The following list shows the configuration parameters for the Web Agent in alphabetical order:

AcceptTPCookie

Allows the Web Agent to accept session (SMSESSION) cookies created by third-party (non-SiteMinder) Web Agents. Third-party agents generate and read SMSESSION cookies using the SiteMinder SDK.

Default: No default

More information:

Configure Support for SDK Third-Party Cookies (see page 100)

AgentConfigObject

Defines the name of an Agent Configuration Object (stored on a policy server) in a local agent configuration file. This parameter is *not* used in Agent Configuration Objects.

Default: no default

Note: If you change the value of this parameter, you must restart the web server to apply the change.

More information:

Parameters Requiring a Server Restart when Changed (see page 25)
WebAgent.conf file for Framework Agents (see page 32)
Parameters Found Only in Local Configuration Files (see page 34)

AgentName

Defines the identity of the Web Agent. It establishes a mapping between the name and the IP address of each web server instance hosting an Agent.

If a value is not set for this parameter, or if the Web Agent does not find a match among the values listed, the Web Agent uses the value set in the DefaultAgentName parameter instead.

Note: This parameter can have more than one value. Use the multi-value option when setting this parameter in an Agent Configuration Object. For local configuration files, add the parameter name followed by each value to separate lines in the file.

Default: No default

Limits: Must contain 7-bit ASCII characters in the range of 32-127, and include one or more printable characters. Cannot contain the ampersand (&) and asterisk (*) characters. Not case-sensitive. For example, the names MyAgent and myagent are treated the same.

Example: myagent1,192.168.0.0

Example: myagent, www.sitea.com

More information:

Set the Agent Name and Default Agent Name Identities (see page 48)

AgentNamesAreFQHostNames

Enables the use of fully qualified host names of a target URL as Web Agent names with FCCs and SCCs.

Default: No

More information:

Configure Credential Collectors in a Mixed Environment (see page 235)
Use FCCs and NTCs in a Mixed Environment (see page 236)
Use SCCs in a Mixed Environment (see page 239)
Enable FCCs/SCCs to Use Agent Names as Fully Qualified Host Names (see page 243)

AgentWaitTime

Specifies the number of seconds that the Web Agent waits for the Low-level Agent Worker process (LLAWP) to become available. When the interval expires the Web Agent tries to connect to the Policy Server.

Setting this parameter can help to resolve agent start-up errors related to LLAWP connections. We recommend starting with the default value and then increasing the interval by 5 seconds at a time until the agent starts successfully.
If you do not want to set this parameter in the Agent Configuration Object or LocalConfig.conf file, you can also set it in the WebAgent.conf file instead.

Default: 5

Example: If you have primary and secondary policy servers, try starting with value from 30 through 40.

Limit: None

Note: This parameter was originally developed for IIS Web Agents, but you can use it with other Framework Agents (on supported Windows platforms) if you experience network latency issues.

More information:

Accommodate Network Latency (see page 53)

AllowCacheHeaders

Specifies whether the Web Agent will remove the cache-related headers from requests for protected resources.

The settings of this parameter affect the following parameters:

- LogOffURI

Default: No

Limits: Yes, No, None

More information:

Control How HTTP Header Resources are Cached (see page 132)

AllowLocalConfig

Instructs the Agent Configuration Object on the Policy Server to read the local configuration file to obtain configuration parameters for the Web Agent. This parameter is used only in Agent Configuration Objects.

You can also add multiple values for this parameter in the Agent Configuration Object to control which parameters can be changed in a local configuration file. When multiple values are set for this parameter, they are processed in the following order:

- If the value of yes exists together with other configuration parameters in the same Agent Configuration Object, yes takes precedence. The other configuration parameters are the only ones that can be changed in the local configuration file.
- If the value no exists together with other configuration parameters in the same Agent Configuration Object, no takes precedence. This lets you quickly disable local configuration entirely without having to remove any of the other configuration parameters from the Agent Configuration Object.
- If the multiple values of yes and no exist in the same Agent Configuration Object, no takes precedence. This lets you quickly disable local configuration entirely without having to remove any of the other configuration parameters from the Agent Configuration Object.

Default: No

Example: yes, EnableAuditing, EnableMonitoring (allows local control of the only the two previous parameters).

More information:

Implement Local Configuration (see page 36)
Restrict Changes to Local Configuration Parameters (see page 38)
Central and Local Configuration Together (see page 39)

AppendIISServerLog

Instructs the Web Agent to add the authenticated user name and SiteMinder transaction ID to the IIS server log on a separate line.

Default: No

Note: This parameter applies to IIS 6.0 Web Agents only.

More information:

Record the User Name and Transaction ID in IIS 6.0 Server Logs (see page 160)

More information:

BadCSSChars

Specifies the URL characters that a Web Agent interprets as a possible cross-site scripting attack.

Default: <,',>

Limits:

- You can specify characters literally or enter the URL-encoded form of that character. For example, you can enter the letter a, or you can enter the encoded equivalent of %61.
- You can specify a maximum number of 4096 characters (including commas used for separating characters).
- You can specify ranges of characters separated by a hyphen. The syntax is: *starting_character-ending_character*. For example, you can enter a-z as a range of characters.
- Specify quotes (") with the URL-encoded equivalent of %22. Do *not* use ASCII.

More information:

Specify Bad URL Characters (see page 219)
Protect Web Sites Against Cross-Site Scripting (see page 225)
Override the Default CSS Character Set (see page 226)

BadFormChars

Specifies the characters that the Web Agent encodes as literal HTML characters before using them as output on a form.

Default: <, >, &, %22

Limits:

- You can specify characters literally or enter the URL-encoded form of that character. For example, you can enter the letter a, or you can enter the encoded equivalent of %61.
- You can specify a maximum number of 4096 characters (including commas used for separating characters).
- You can specify ranges of characters separated by a hyphen. The syntax is: *starting_character-ending_character*. For example, you can enter a-z as a range of characters.
- Specify quotes (") with the URL-encoded equivalent of %22. Do *not* use ASCII.

More information:

Specify Bad URL Characters (see page 219)
Specify Bad Form Characters (see page 221)

BadQueryChars

Specifies characters that the Web Agent prohibits in the query string portion (following the '?') in a URL.

Default: Empty (any characters allowed in query strings)

Limits:

- You can specify characters literally or enter the URL-encoded form of that character. For example, you can enter the letter a, or you can enter the encoded equivalent of %61.
- You can specify a maximum number of 4096 characters (including commas used for separating characters).
- You can specify ranges of characters separated by a hyphen. The syntax is: *starting_character-ending_character*. For example, you can enter a-z as a range of characters.
- Specify quotes (") with the URL-encoded equivalent of %22. Do *not* use ASCII.

Example: %25 blocks URL-encoded characters in queries.

More information:

Specify Bad Query Characters (see page 222)
Specify Bad Form Characters (see page 221)

BadUrlChars

Specifies the character sequences that cannot be used in URL requests. The Web Agent checks the characters in the URL that occur before the "?" character against those specified by this parameter. If any of the specified characters are found, the Web Agent rejects the request.

You can specify the following characters:

- a backward slash (\)
- two forward slashes (//)
- period and a forward slash (./)
- forward slash and a period (/.)
- forward slash and an asterisk (/*)
- an asterisk and a period (*.)
- a tilde (**~**)
- %2d
- %20
- %00-%1f
- %7f-%ff
- %25

Separate multiple characters with commas. Do *not* use spaces.

You can use the bad URL characters in CGI parameters if the question mark (?) precedes the bad URL characters.

Default: <,>,&,;

Limits:

- You can specify characters literally or enter the URL-encoded form of that character. For example, you can enter the letter a, or you can enter the encoded equivalent of %61.
- You can specify a maximum number of 4096 characters (including commas used for separating characters).
- You can specify ranges of characters separated by a hyphen. The syntax is: *starting_character-ending_character*. For example, you can enter a-z as a range of characters.

- Specify quotes (") with the URL-encoded equivalent of %22. Do *not* use ASCII.

More information:

Specify Bad URL Characters (see page 219)
IIS 6.0 Servers and BadURLChars Settings (see page 220)

CacheAnonymous

Specifies if the Web Agent caches anonymous user information. You may want to set this parameter in any of the following situations:

- If your web site gets mostly anonymous users and you want to store their session information.
- If your web site gets a mix of registered and anonymous users.

  You may want to disable this parameter to keep the anonymous user information from filling the cache and leaving no room for registered users.

Default: No

Note: If you change the value of this parameter, you must restart the web server to apply the change.

More information:

Cache Anonymous Users (see page 185)

CCCExt

Specifies the MIME type for the Cookie Provider credential collector.

This parameter affects the following parameters:

- CookieProvider

Default: .ccc

More information:

Specify the Cookie Provider (see page 86)
Set Up Credential Collectors for IIS and Domino Web Servers (see page 233)

ConformToRFC2047

Indicates whether the Web Agent conforms to RFC 2047. If this parameter is missing, the Web Agent follows the default behavior.

Default: Yes

More information:

Disable Conformance to RFC 2047 (see page 135)

ConstructFullPwsvcUrl

Instructs the Web Agent to generate a URL with a fully qualified domain name for redirecting users to the Password Services application. This lets you host the Password Services application on a particular web server. The Web Agent generates a URL that resembles the following example:

HTTP://my.server.com:80/path/to/passwordservices.cgi

If a fully-qualified URL is *not* used, the Web Agent assumes that the Password Services application is hosted on the same web server and uses a relative URL for redirects.

Default: No

More information:

Use a Fully Qualified URL for Password Services Redirects (see page 255)

CookieDomain

Defines the cookie domain of the Web Agent that you specified during the Web Agent installation. This must be a fully qualified domain name with at least two periods. For example, the .myorg.com cookie domain matches the following servers:

- w1.myorg.com
- w2.myorg.com
- w3.sales.myorg.com

All web servers in this domain can exchange cookies with a user's browser. Servers in the same cookie domain use cookies to verify a user's credentials.

Default: Empty

Example: .mycompany.com

Note: This value is case-sensitive.

More information:

How to Configure Single Sign-On (see page 83)
Specify the Cookie Domain (see page 85)
Implement Cookie Domain Resolution (see page 102)
Modify the Cookie Domain (see page 105)

CookieDomainScope

Specifies the number of sections (areas separated by a period) in the domain name.

Default: 0

Example: For a domain named server.division.myorg.com, in a cookie domain of division.myorg.com, set the CookieDomainScope to 3.

More information:

Implement Cookie Domain Resolution (see page 102)
Specify the Cookie Domain (see page 85)

CookiePath

Specifies the cookie path for the following secondary agent browser cookies:

- xxSESSION
- xxIDENTITY
- xxDOMINODATA
- xxCHALLENGE (including SSL_CHALLENGE_DONE)
- xxDATA
- xxSAVEDSESSION

For example, setting this parameter to /BasicAuth, all of the secondary agents in the previous list are created using /BasicAuth as the path. If not specified, the default value is used.

The CookiePath is *not* added to credential cookies (such as xxxxCRED) to maintain backwards compatibility with 4.x agents.

The following cookies will *always* use the root (/) path:

- ONDENIEDREDIR
- TRYNO

If the CookiePathScope parameter is greater than zero, the CookiePath parameter settings are overriden.

Default: / (root)

More information:

Specify the Cookie Path for Agent Cookies (see page 111)
Configure Full Logoff (see page 107)
Configure Full Logoff Support for Domino Agents (see page 277)

CookiePathScope

Specifies the scope of the cookie path for the following secondary agent cookies:

- xxSESSION
- xxIDENTITY
- xxDOMINODATA
- xxCHALLENGE (including SSL_CHALLENGE_DONE)
- xxDATA
- xxSAVEDSESSION

Using a CookiePathScope greater than zero in this parameter overrides the setting of the CookiePath parameter.

Default: 0

More information:

Specify the Cookie Path for Agent Cookies (see page 111)

CookieProvider

Specifies the URL (using the fully qualified domain name) of the web server where the Web Agent that is acting as the cookie provider resides. The cookie provider name must have a .ccc extension.

- For IIS, Sun Java System and Domino web servers, the URL syntax is as follows:

  http://*server.domain:port*/siteminderagent/SmMakeCookie.ccc

- For Apache and Apache-based web servers, the URL syntax is as follows:

  http://*server.domain:port*/SmMakeCookie.ccc

This parameter affects the following parameters:

- CCCExt
- SessionUpdatePeriod

Default: No default

Example: (IIS, Sun Java System and Domino web servers)
http://server1.myorg.com:80/siteminderagent/SmMakeCookie.ccc

Example: (Apache and Apache-based web servers)
http://server1.myorg.com:80/SmMakeCookie.ccc

More information:

Single Sign-On Across Multiple Cookie Domains (see page 78)
How to Configure Single Sign-On (see page 83)
Specify the Cookie Provider (see page 86)
Modify the Session Update Period (see page 87)

CookieValidationPeriod

Specifies the time period (in seconds) in which the receiving agent will accept the session cookie. After this time passes, the session cookie will not be accepted. If this field is not used or is set to zero, the session cookie expires when the Idle Timeout and Max Session Timeout values are met.

Default: Empty

More information:

Protect Session Cookies from Misuse with Validation Periods and Expired Cookie URLs (see page 99)

CSSChecking

Specifies whether the Web Agent checks URLs (including any query string) for escaped and unescaped characters (as defined by the list in the BadCSSChars parameter) that may be part of an executable script.

Default: Yes

More information:

Configure the Web Agent to Check For Cross Site-Scripting (see page 226)

CSSErrorFile

Specifies the location of a custom-error message file or URL that you want to display to the users if they try to open a URL that contains possible cross-site scripting characters.

Default: No default

More information:

Custom Error Handling For Applications (see page 139)
How to Set Up Error Handling (see page 141)

Custom401ErrorFile

Specifies the customized HTML page to display when users receive a 401 (insufficient privileges) browser error.

Default: No default

More information:

Custom Error Handling For Applications (see page 139)
Notes for Custom 401 Pages (see page 140)
How to Set Up Error Handling (see page 141)

CustomIpHeader

Specifies an HTTP header that the Web Agent should look for to find the requestor's IP address. If no value is specified for this parameter, the default is an empty string. No maximum length is enforced and the value may be any string that contains a valid HTTP header value, for example, HTTP_ORIGINAL_IP.

Default: No default

More information:

Configure IP Address Validation (see page 131)

DecodeQueryData

Specifies whether the Web Agent decodes the query data in a URL before calling the Policy Server. Set this parameter to yes if you need do any of the following tasks in your environment:

- If you need to ensure the rules filer acts against the proper string.
- If you need to or write rules against the data in a query string.

Default: No

More information:

Decode Query Data in a URL (see page 205)

DefaultAgentName

Defines a name that the Web Agent uses when it receives a request on an IP address or interface for which there is no agent name specified in the AgentName parameter.

If you are using virtual servers, you can set up your SiteMinder environment quickly by using a DefaultAgentName instead of defining a separate Web Agent for each virtual server.

Important! If you do not specify a value for the DefaultAgentName parameter, you must list every agent identity in the AgentName parameter. Otherwise, the Policy Server will not be able to tie policies to the Web Agent.

Default: No default

Limits: Must contain 7-bit ASCII characters in the range of 32-127, and include one or more printable characters. Cannot contain the ampersand (&) and asterisk (*) characters. Not case-sensitive. For example, the names MyAgent and myagent are treated the same.

More information:

Set the Agent Name and Default Agent Name Identities (see page 48)
Assign Web Agent Identities for Virtual Servers (see page 70)
Configure Credential Collectors in a Mixed Environment (see page 235)
Use FCCs and NTCs in a Mixed Environment (see page 236)
Use SCCs in a Mixed Environment (see page 239)

DefaultPassword

Specifies a default password for the associated Windows user that is used to access IIS resources as a proxy user.

Important! If you want to encrypt this parameter, set it centrally in the Agent Configuration Object. If this parameter is set in a local configuration file, it will not be encrypted and will be less secure.

Default: No default

Note: This parameter applies to IIS Web Agents only.

More information:

Use an IIS Proxy User Account (IIS Only) (see page 144)
How the IIS Web Agent Populates the REMOTE_USER Variable (see page 119)

DefaultUsername

Specifies the name of a Windows user that is used to access IIS resources as a proxy user. When users want to access resources on an IIS web server protected by SiteMinder, they may not have the necessary server access privileges. For example, if users are stored in an LDAP user directory on a UNIX system, those users may not have access to the Windows system with the IIS web server.

The Web Agent must use this NT user account, which is assigned by an NT administrator, to act as a proxy user account for users granted access by SiteMinder.

Default: No default

Note: This parameter applies to IIS Web Agents only.

More information:

Use an IIS Proxy User Account (IIS Only) (see page 144)
How the IIS Web Agent Populates the REMOTE_USER Variable (see page 119)

DeleteCerts

Specifies if the certificates stored on a Stronghold server will be removed when the Web Agent finishes using them.

Default: No

Note: This parameter applies to Apache Web Agents only.

More information:

Delete Certificates from Stronghold (Apache Agent Only) (see page 280)

DisableAuthSrcVars

Specifies whether the Web Agent disables the following default SiteMinder authentication source HTTP header variables:

- SM_AUTHDIRNAME
- SM_AUTHDIRSERVER
- SM_AUTHDIRNAMESPACE
- SM_AUTHDIROID

Note: You cannot disable individual variables. You can only disable a category of several variables.

Default: No

More information:

Disable Default HTTP Header Variables (see page 125)

DisableDirectoryList

Specifies whether the Web Agent allows a user to view or browse the contents of a directory without challenging them first. This occurs when *all* of the following conditions are true:

- The realm is set to protect a root resource (/)
- The default web page in the directory (such as index.html) is renamed or deleted.

Default: No

Note: This parameter applies to Sun Java System Agents only.

More information:

Restrict Directory Browsing on a Sun Java System Server (see page 63)

DisableDNSLookups

Prevents the Web Agent from performing DNS lookups.

Important! Fully qualified domain names must be used for cookie-based functions to work properly when the value of this parameter is set to yes.

More information:

Help Prevent DNS DOS Attacks (see page 228)

DisableDotDotRule

Specifies whether the Web Agent blocks access to a URL that contains two periods separated by a slash (/).

The settings of this parameter affect the following parameter:

- IgnoreExt

Default: No (the rule is applied)

More information:

Handle Complex URIs (see page 217)

DisablePostDataLimit

Specifies whether a Web Agent observes the 64 KB data-size limit when preserving or filtering POST data. This does not affect the standard POST operation, but it does affect the following:

- POST preservation
- eTelligent Rules Post variables
- TransactionMinder authentication schemes

Default: No (limit enforced)

Important! Change this parameter to yes at your own risk.

Note: This parameter applies to IIS 5.0 Web Agents only.

More information:

Limit Size of Post Data (IIS 5.0 Agents only) (see page 58)

DisableSessionVars

Specifies whether the Web Agent disables the following default SiteMinder user session HTTP header variables:

- SM_SERVERSESSIONID
- SM_SERVERSESSIONSPEC
- SM_SERVERIDENTITYSPEC
- SM_SESSIONDRIFT
- SM_TIMETOEXPIRE

Note: You cannot disable individual variables. You can only disable a category of several variables.

Default: No

More information:

Disable Default HTTP Header Variables (see page 125)

DisableUserNameVars

Specifies whether the Web Agent disables the following default SiteMinder user name HTTP header variables:

- SM_USER
- SM_USERDN
- SM_DOMINOCN

Note: You cannot disable individual variables. You can only disable a category of several variables.

Default: No

More information:

Disable Default HTTP Header Variables (see page 125)

DominoDefaultUser

Specifies the name by which the Domino Web Agent identifies the users that SiteMinder has previously authenticated against another directory to the Domino server.

Important! This parameter must be encrypted if it is stored in a local configuration file. Use the encryptkey tool to encrypt this parameter. Do not change it by editing the local configuration file directly.

Default: No default

Note: This parameter applies to Domino Web Agents only.

More information:

Authenticate Users with the Domino Server (see page 262)
Authenticate as the Actual User or the Default User (see page 265)
Modify the Domino Default User and the Domino Super User (see page 266)
Use Encryptkey to Set the Domino Default or Super User (see page 267)

DominoLegacyDocumentSupport

Specifies how a Web Agent handles user requests for protected Lotus Notes documents in a Domino environment. Setting this parameter to yes grants users ReadForm permission only for the requested document.

Default: No

Note: This parameter applies to Domino Web Agents only.

More information:

Control Access to Lotus Notes Documents (see page 271)

DominoLookUpHeaderForLogin

Instructs the Domino Web Agent to ask the Domino web Server if the user requesting access to a resource is unique or ambiguous within the Domino user directory. This helps if a user requesting access to a resource has the same name as other users in the user directory.

Default: No

Note: This parameter applies to Domino Web Agents only.

More information:

Use a SiteMinder Header for Authentication (see page 268)

DominoMapUrlForRedirect

Instructs the Web Agent to map (normalize) the URL from the Domino server representation to a URL-friendly name for the redirect to the Forms Credential Collector (FCC). The FCC can process the request for the requested Domino resource. If this parameter is missing, the default behavior occurs. If this parameter is set to no, the Web Agent does not map the URL, and performs FCC redirects using the original Domino server representation.

The DominoNormalizeUrls parameter must also be set to yes, otherwise the URL will not be normalized.

Default: Yes

Note: This parameter applies to Domino Web Agents only.

More information:

Map URLs for FCC Redirects (see page 267)

DominoNormalizeUrls

Specifies if the SiteMinder Web Agent converts Domino URLs to a URL-friendly name before redirecting them to a Forms Credential Collector.

The MapUrlsForRedirect parameter must also be set to yes for the Domino URLs to be converted.

If the DominoNormalizeUrls parameter is set to no, URLs will *not* be normalized, even if the MapUrlsForRedirect parameter is set to yes.

Important! If you set the DominoNormalizeUrls parameter to no, you cannot protect individual documents within a Notes database; you can only protect the entire database or subdirectories of the Domino Web server.

Default: Yes

Note: This parameter applies to Domino Web Agents only.

More information:

Map URLs for FCC Redirects with a Domino Web Agent (see page 269)
Disable URL Normailization (see page 270)

DominoSuperUser

Identifies a user who has access to all resources on the Domino server, and ensures that all users successfully logged into SiteMinder will be logged into Domino as the Domino SuperUser.

This value can be encrypted.

This parameter affects the following parameters:

- SkipDominoAuth

Default: No default

Note: This parameter applies to Domino Web Agents only.

More information:

Authenticate Users with the Domino Server (see page 262)
Authenticate as the Domino Super User (see page 265)
Modify the Domino Default User and the Domino Super User (see page 266)
Use Encryptkey to Set the Domino Default or Super User (see page 267)

DominoUseHeaderForLogin

Instructs the Domino Web Agent to pass the SiteMinder header value to the Domino web server. The Domino server uses the header data to identify a user in its user directory.

Default: No default

Note: This parameter applies to Domino Web Agents only.

More information:

Use a SiteMinder Header for Authentication (see page 268)

EnableAuditing

Specifies whether the Web Agent logs all successful authorizations that are stored in the user session cache. When enabled, user authorizations are logged even when the Web Agent uses information from its cache instead of contacting the Policy Server. Web Agents log user names and access information in native web server log files when users access resources.

Default: No

More information:

Track User Activities or Application Usage with Auditing (see page 157)

EnableFormCache

Controls the forms template cache. Setting this parameter to yes, improves the performance of forms authentication. To disable the cache, set this parameter to no.

Default: Yes

More information:

Configure the Form Cache (see page 241)

EnableIntroscopeApiSupport

Collects information about the SiteMinder Web Agent and sends it to CA Wily Introscope using a plug-in. This parameter uses the following settings:

- When set to yes, the Wily plug-in calls an API to collect the data.
- When set to no, the Wily plug-in creates an HTTP header with the data.
- When set to both, the Wily plug-in calls the API *and* creates an HTTP header with the data.
- When set to none, data not collected.

Default: No.

Limits: Yes, Both, No, None.

Example: (HTTP header) sm-wa-perf-counters = *server_name*.example.com:6180,86117203,86118343,1,0,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,1125,0,15,1,1,750,750,

More information:

Use CA Wily Introscope to Monitor Web Agents (see page 51)

EnableMonitoring

Specifies whether the SiteMinder Web Agent sends monitoring information to the Policy Server.

Default: No

More information:

Monitor Web Agents with the OneView Monitor (see page 52)

EnableOtherAuthTrans

Allows the use of other AuthTrans functions along with SiteMinder.

Default: No

Note: This parameter applies to Sun Java System Agents only.

More information:

Handle Multiple AuthTrans Functions (Sun Java System only) (see page 138)

EnableWebAgent

Activates a Web Agent and allows it to communicate with the Policy server. Set this parameter to yes only after you have finished changing all of the configuration parameters.

Default: No

Note: This parameter is used in local configuration files only.

More information:

Enable a Web Agent (see page 65)
Disable a Web Agent (see page 66)
WebAgent.conf file for Framework Agents (see page 32)
Parameters Found Only in Local Configuration Files (see page 34)

EncryptAgentName

Controls whether the agent name is encrypted when the Web Agents adds its name to the URL that redirects a user to a forms, SSL, or NTLM credential collector. It also controls whether the credential collector decrypts the name when it receives the URL.

Default: Yes

More information:

Encrypt the Agent Name (see page 50)
Configure the FCC to Use a Single Resource Target (see page 240)

EnforceRealmTimeouts

Determines if the Web Agent overrides the session time-out values from the first realm a user accesses with the session time-out values from a subsequent realm a user accesses using single sign-on. When this parameter is set to yes, the Web Agent looks at the session time-outs returned during an SMSESSION cookie validation and honors the time-out values for the subsequent realm in which the user is being validated. When this parameter is set to no, the Agent honors the time-outs of the original login session. When a user moves to a new realm, the Web Agent enforces the idle or session time-outs from the first realm and not the timeouts from the subsequent realm.

Default: No

More information:

How to Enforce Timeouts across Multiple Realms (see page 97)

ExpiredCookieURL

(Optional) Specifies a URL that the agent redirects the user to after any session cookie has expired. If neither the born date nor the CookieValidationPeriod are configured, the agent ignores the settings and processes the cookie as usual (backwards compatibility).

More information:

Protect Session Cookies from Misuse with Validation Periods and Expired Cookie URLs (see page 99)

ExpireForProxy

Prevents a forward proxy server from caching content (pages and potentially headers or cookies). When this parameter is set to yes (enabled), the Web Agent inserts an Expires or Cache-control header into the HTTP response. If content is not cached, subsequent requests continue to be forwarded.

When the ExpireForProxy parameter is set to yes, the Web Agent inserts the strings specified in the appropriate ProxyHeaders*suffix* parameter into the HTTP response based upon what type of request was performed.

The Web Agent adds strings into the HTTP responses as follows:

- For HTTP/1.1 requests—if the resource is auto-authorized, then the Web Agent inserts the value of all ProxyHeadersAutoAuth parameters into the HTTP response. If the resource is protected, then the Web Agent inserts the value of all ProxyHeadersProtected parameters into the HTTP response. If the resource is not protected, then the Web Agent inserts the value of all ProxyHeadersUnprotected parameters into the HTTP response.
- For HTTP/1.0 requests—if the resource is auto-authorized, then the Web Agent inserts the value of all ProxyHeadersAutoAuth10 parameters into the HTTP response. If the resource is protected, then the Web Agent inserts the value of all ProxyHeadersProtected10 parameters into the HTTP Response. If the resource is not protected, then the Web Agent inserts the value of all ProxyHeadersUnprotected10 parameters into the HTTP response.

Default: No

Note: This parameter applies to proxy servers only.

More information:

Configure Agents that Sit behind Proxy Servers (see page 188)
Customize the Cache-Control and ExpireForProxy Header Settings (see page 190)
Proxy Header Usage Notes (see page 192)

FCCCompatMode

Enable an FCC/NTC to serve up forms for resources protected by 4.x Web Agents or third party applications.

Note: SMUSRMSG is supported for the custom authentication scheme only when FCCCompatMode set to yes.
Default: (traditional agents) Yes

Default: (framework agents) No

Important! Setting this parameter to no removes support for version 4.x of the Netscape browser.

More information:

Use SM_AGENT_ATTR_USRMSG Response for a Forms Challenge (see page 135)
Query String Encryption of Redirect URLs (see page 207)
Use FCCs and NTCs in a Mixed Environment (see page 236)

FCCExt

Specifies a MIME value for credential collectors on IIS or Domino web servers.

Default: .fcc

More information:

Set Up Credential Collectors for IIS and Domino Web Servers (see page 233)

FCCForceIsProtected

Specifies whether the Web Agent makes an additional IsProtected call to the Policy Server to establish a realm context so that the Web Agent can log a user in to access a protected resource.

When this parameter is set to no, the Web Agent uses the realm information obtained from its initial IsProtected call to the Policy Server instead.

Default: Yes

More information:

Disable FCC Realm Context Confirmation to Improve Performance (see page 242)

ForceCookieDomain

Forces the Web Agent to append its cookie domain to the host name in a URL request that does not specify a domain or contains only an IP address. This parameter works together with the ForceFQHost parameter for added functionality.

Default: No

More information:

Force the Cookie Domain (see page 101)
Force Fully Qualified Domain Names (see page 104)

ForceFQHost

Forces a Web Agent to use a fully qualified domain name. This parameter uses configured domain name system (DNS) services to force the appending of the cookie domain to the host name in a URL request through DNS services and not an Agent. If the Web Agent receives a request that contains a partial URL, the Web Agent redirects the request back to the same destination resource specified in the original URI. The redirect request uses the fully qualified host name, which the Web Agent determines using the configured DNS services. Use this parameter with the ForceCookieDomain parameter for added functionality.

Default: No

Example: When the Web Agent receives a request from http://host1/page.html, it responds with http://host1.myorg.com/page.html. If the Web Agent receives a request such as http://123.113.12.1/page.html, it responds with http://host1.myorg.com/page.html.

Note: These examples work only if the proper DNS lookup tables are set up. If a partial domain is entered, the result depends on whether or not the DNS lookup can resolve it. If the request resolves as an invalid host, an error will result. Most likely, such a request would not even reach the web server.

More information:

Force the Cookie Domain (see page 101)
Force Fully Qualified Domain Names (see page 104)

ForceIISProxyUser

Specifies whether the Web Agent uses an IIS proxy account to grant access to requested resources on IIS web servers to users who normally lack sufficient privileges to access the IIS web server.

This parameter affects the following parameters:

- DefaultUserName
- DefaultPassword

Default: No

Note: This parameter applies to IIS Web Agents only.

More information:

How the IIS Web Agent Populates the REMOTE_USER Variable (see page 119)
Use an IIS Proxy User Account (IIS Only) (see page 144)

FormCacheTimeout

Specifies the number of seconds that an object may reside in cache before being considered invalid. When the timeout interval expires, the date and time of the form template file is compared against the time that the cache object was created. If the object in cache is stored more recently than the file on the system's disk, the timeout is reset for another interval; otherwise, the object is removed from cache.

Default: 600

More information:

Configure the Form Cache (see page 241)

GetPortFromHeaders

Directs the Web Agent to obtain the port number from the HTTP HOST request header instead of obtaining it from the web server service structures.

Default: No

Note: This parameter is required for Apache Web Agents.

More information:

Use the HTTP HOST Request for the Port Number (see page 61)

HostConfigFile

Specifies the path to the SMHost.conf file (in an IIS 6.0 or Apache agent) that is created after a trusted host computer has been successfully registered with a Policy server. All Web Agents on a computer share the SMHost.conf file.

Default: No default

Note: If you change this parameter, you must restart the web server to apply the change.

More information:

WebAgent.conf file for Framework Agents (see page 32)
Parameters Requiring a Server Restart when Changed (see page 25)
Parameters Found Only in Local Configuration Files (see page 34)

HTTPHeaderEncodingSpec

Selects the specification that the Web Agent uses for encoding the HTTP header values and all custom HTTP-COOKIE responses. The value for this parameter uses the following syntax:

encoding_spec, *wrapping_spec*

Including the wrapping specification (RFC-2047) is optional, but we recommend using it.

Default: No default (if left blank, the Web Agent uses UTF-8 encoding with no wrapping)

Example: Shift-JIS,RFC-2047

More information:

Set the HTTP Header Encoding Spec (see page 134)

HttpsPorts

Specifies the secure ports the Web Agent listens on if you are using an SSL connection to the web server. If you specify a value for this parameter, you must include all the ports for all the web servers that serve secure requests. If you do not specify a value, the Web Agent reads the HTTP scheme from the web server's context.

If a server is behind an HTTPS accelerator (which converts HTTPS to HTTP), the requests are treated as SSL connections by your browser.

Default: Empty

Example: 80

Example: (multiple ports) 80,8080,8083

More information:

Define HTTPS Ports (see page 137)
SiteMinder Reverse Proxy Deployment Considerations (see page 198)

IdleTimeoutURL

Specifies the URL where the Web Agent should redirect the user when the idle time-out for the session occurs.

Example:
http://example.mycompany.com/sessionidletimeoutpage.html

Note: IdleTimeoutURL should only be used for non-persistent sessions; it has no effect if configured for persistent sessions.

More information:

Redirect a User after a Session Time-out (see page 98)

IgnoreCPForNotprotected

Prevents the cookie provider from being queried for unprotected resource requests. When this parameter is set to no, all requests are directed to the cookie provider by the Web Agent. For traditional (non-framework) Agents, a cookie provider must be configured for the value of this parameter to appear in the Web Agent log file.

Default: No

More information:

Ignore the Cookie Provider for Unprotected Resources (see page 100)
Parameters Not Used by Framework Agents (see page 24)

IgnoreExt

Specifies the types of resources for which the Web Agent passes requests to the web server without checking SiteMinder policies.The Web Agent allows access to the items specified by this parameter even if they exist in a realm that is protected by a SiteMinder policy.

Requests for resources that meet either of the following conditions may be ignored:

- The resource ends in one of the extensions that you configure the Web Agent to ignore.
- The URI of the protected resource contains a single period (.).

  For example, if a URI for a requested resource is /my.dir/ the Web Agent passes the request directly to the web server.

Default: .class, .gif, .jpg, .jpeg, .png, .fcc, .scc, .sfcc, .ccc, .ntc

Important! Use caution when setting the IgnoreExt parameter. There are some security issues that you may want to consider.

More information:

Specify Virtual Servers for the Web Agent to Ignore (see page 71)
Control How HTTP Header Resources are Cached (see page 132)
Reduce Overhead by Ignoring File Extensions of Unprotected Resources (see page 214)
Handle Complex URIs (see page 217)

IgnoreHost

Specifies the fully qualified domain names of any virtual servers that you want the web Agent to ignore. Resources on such virtual servers will be auto-authorized, and the Web Agent always grants access to them regardless of which client makes the request. The authorization decision is based on the configuration of the Web Agent instead of being based on a policy.

The list of ignored hosts is checked first before any other auto-authorization checks, such as the IgnoreExt and IgnoreURL settings. Therefore, the double-dot rule will not trigger an authorization call to the Policy Server for resources on an ignored host but would not be ignored by extension.

The host portion of the URL entries for the IgnoreHost parameter must exactly match what the Web Agent reads for the host header of the requested resource.

Note: This value is case-sensitive.

If the URL uses a specific port, then the port must specified.

For centrally-managed agents, use a multi-value parameter in the Agent Configuration Object to represent several servers. For agents configured with a local configuration file, list each host on a separate line in the file.

Example: (URL shown with port specified)

IgnoreHost="myserver.example.org:8080"

Example: (local configuration file)

IgnoreHost="my.host.com"

IgnoreHost="your.host.com"

Default: No default

More information:

Specify Virtual Servers for the Web Agent to Ignore (see page 71)

IgnoreQueryData

Specifies whether the Web Agent will cache the entire URL (including the query strings) and send the entire URI to the Policy Server for rule processing. A full URL string contains a URI, a hook (?), and some query data, as shown in the following example:

URI?*query_data*

URLs that have been the subjects of requests are cached by default. Subsequent requests search the cache for a match. If requests for the same URI contain different query data, the match fails. Ignoring the query data improves performance.

When the IgnoreQueryData parameter is set to yes, the following occurs:

- The URL is truncated at the hook. Only the URI is cached and sent to the Policy Server. The query data is maintained elsewhere, for the purpose of maintaining the proper state for redirects.
- Only the part before the hook is sent to the Policy Server for rule processing.
- Both URIs in the following example are handled as the same resource:

  /myapp?data=1

  /myapp?data=2

When the IgnoreQueryData parameter is set to no, the following occurs:

- The entire URL is cached.
- The entire URI is sent to the Policy Server for rule processing.
- The URIs in the following example are handled as different resources:

  /myapp?data=1

  /myapp?data=2

Default: No

Important! Do not enable this setting if you have policies which depend on URL query data.

More information:

Ignore Query Data in a URL (see page 206)

IgnoreUrl

Specifies a URI within a URL that will not be protected. Users attempting to access the resource associated with the URI will not be challenged. The Web Agent ignores the URI portion of the string after three forward slashes. For example, if you set this parameter to the following value:

http://www.example.com/directory

The Web Agent ignores the following URI:
directory

The Web Agent ignores the specified URI wherever it occurs, even if it is under a different domain. For example, the Web Agent ignores the URI shown previously in all of the following URLs:

http://www.example.com/directory
http://www.example.net/directory
http://www.example.org/directory

Note: This value is case-sensitive.

Default: No default.

Example: (multiple URIs in local configuration file)

IgnoreUrl="http://www.example.com/directory"

IgnoreUrl="http://www.example.com/directory2"

Example: (using a URI only, without specifying a domain)

IgnoreUrl="/*resource*/"

More information:

Allow Un-restricted Access to URIs (see page 210)
Specify Virtual Servers for the Web Agent to Ignore (see page 71)

InsecureServer

Allows Web Agents for IIS 5 to enforce native IIS security mechanisms by providing a Windows user security context. Add this parameter to the Agent Configuration Object or local configuration file with the value you want.

If this parameter is set to yes, the Web Agent stores encrypted credentials in paged memory, which can be written to the operating system's page file and saved to a hard disk.

Important! If your hard disk is stolen or compromised, confidential data could be exposed.

If this parameter is no, the Web Agent stores encrypted credentials in protected kernel memory. This setting is more secure, but it places more demands on the physical memory of your IIS server.

Default: No

Note: This parameter applies to IIS 5.0 Web Agents only. It is not used in Framework Web Agents.

More information:

Store Encrypted Credentials in a Page File (IIS 5.0 Only) (see page 143)
Parameters Not Used by Framework Agents (see page 24)

LegacyCookieProvider

Controls whether a framework agent sends a POST request to a cookie provider. When framework agents send a POST request to a traditional agent that is acting as a cookie provider, the redirected request becomes a GET instead and fails. When set to no, the framework agent sends the POST request to the cookie provider. When set to yes, the framework agent does not send the POST request to the cookie provider.

If you are using central agent configuration, you must add this parameter to your Agent Configuration Object. This parameter already exists in local configuration files.

Default: No (POST requests sent)

Note: This parameter applies to framework agents only.

More information:

Ignore the Cookie Provider for POST Requests (Framework Agents Only) (see page 101)

LegacyEncoding

Forces the Web Agent to replace any dollar sign ($) characters in legacy URLs with a hyphen (-). This also ensures backwards comparability with MSR, Password Services, and DMS. When this parameter is set to no, a Web Agent converts the string SM to -SM-. When this parameter is set to yes, the Web Agent does *not* convert the dollar sign ($) character.

Default: (Framework Agents) No

Default: (Traditional Agents) Yes

More information:

Accommodate Legacy URL Encoding (see page 280)
Query String Encryption of Redirect URLs (see page 207)

More information:

Enable Post Preservation between Framework and Traditional Agents (see page 245)

LegacyTransferEncodingBehavior

Specifies the type of message encoding used by the Web Agent. When the value of this parameter is set to no, transfer-encoding is supported.

When the value of this parameter is set to yes, content encoding is used. The transfer-encoding header is ignored and only the content-length header is supported.

Default: No

- **Important!** If you set the value of this parameter to yes, these features will not work: Federation; preservation of POST data longer than 4 KB; and large certificates may not be recognized.

Note: This parameter applies to Apache Web Agents only.

More information:

Use Legacy Applications with an Apache Web Agent (see page 61)

LegacyVariables

Specifies if the Web Agent uses underscores in HTTP header names. With some web servers (such as the Sun Java System), using the underscore character in the HTTP headers causes problems with some applications.

When this parameter is set to no, the HTTP headers will not have underscores, as shown in the following example:

SM*HeaderName*

When this parameter is set to yes, the HTTP headers will use underscores, as shown in the following example:

SM_*HeaderName*

Default: (traditional agents) Yes

Default: (framework agents) No

More information:

Enable Legacy Variables for HTTP Headers (see page 137)

LoadPlugin

Specifies which plug-ins are loaded for IIS 6.0 and Apache 2.0 Web Agents. The plug-ins support different types of Agent functions.

Default: No default

Important! Do not add any other parameters to the WebAgent.conf file.

The following plug-ins are available:

HttpPlugin

Specifies whether the Web Agent operates as an HTTP agent.

Default: Enabled

SAMLAffiliatePlugin

Allows communication between the Web Agent and a SAML Affiliate Agent (if you have purchased Federation Security Services).

Default: Disabled

Affiliate10Plugin

Allows communication between the Web Agent and a 4.x Affiliate Agent. This is not used by the SAML Affiliate Agent.

Default: Disabled

More information:

WebAgent.conf file for Framework Agents (see page 32)

localconfigfile

Specifies the location of the LocalConfig.conf file, which contains most of the Agent configuration settings.

Default: No default

More information:

WebAgent.conf file for Framework Agents (see page 32)

LogAppend

Adds new log information to the end of an existing log file. When this parameter is set to no, the entire log file is rewritten each time logging is invoked.

Default: No

More information:

Set Up and Enable Error Logging (see page 164)

LogFile

Specifies whether the Web Agent records logs. If this parameter is set to yes in a local configuration file, logging is enabled even if the AllowLocalConfig parameter of an Agent Configuration Object is set to no.

Default: No

More information:

Set Up and Enable Error Logging (see page 164)

LogFileName

Specifies the full path (including the file name) of the log file.

Default: No

Example: /export/iPlanet/servers/https-jsmith/logs/WebAgent.log

More information:

Set Up and Enable Error Logging (see page 164)

LogFileSize

Specifies the size limit of the log file in megabytes. When the current log file reaches this limit, a new log file is created. The new log file uses one of the following naming conventions:

- For framework agents, the new log file has a sequence number appended to the original name. For example, a log file named myfile.log is renamed to myfile.log.1 when the size limit is reached.
- For traditional agents, the new log files are named by appending the date and timestamp to the original name. For example, a log file named myfile.log, is renamed to myfile.log.09-18-2003-16-07-07 when the size limit is reached.

You must archive or remove the old files manually.

Note: Rolling logs are *not* supported for Apache 1.x web servers on UNIX systems. Accept the default or leave this setting blank.

Default: 0 (no rollover)

Example: 80

More information:

Set Up and Enable Error Logging (see page 164)

LogFilesToKeep

Specifies the number of Web Agent log files that are kept. New log files are created in the following situations:

- When the Web Agent starts.
- When the size limit of the log file (specified by the value of the LogFileSize parameter) is reached.

Changing the value of this parameter does *not* automatically delete any existing logs files which exceed the number that you want to keep. For example, If your system has 500 log files stored, and you decide to keep only 50 of those files, the Web Agent does *not* delete the other 450 files.

Setting the value of this parameter to zero retains all the log files.

Default: 0

More Information:

Limit the Number of Log Files Saved (see page 166)

LogLocalTime

Specifies whether the logs use Greenwich Mean Time (GMT) or local time. To use GMT, change this setting to no. If this parameter does not exist, the default setting is used.

Default: Yes

More information:

Set Up and Enable Error Logging (see page 164)

LogOffUri

Enables full log off and specifies the location of a custom web page on your web server that appears to users after they are successfully logged off. You must configure this page so that it cannot be stored in a browser cache. Otherwise, a browser may display a logoff page from its cache without logging the user off. This may give an unauthorized user an opportunity to assume control of a session.

Note: When the CookiePath parameter is set, the value of the LogOffUri parameter must point to the same cookie path. For example, if the value of your CookiePath parameter is set to example.com, then your LogOffUri must point to example.com/logoff.html

Default: No default

Limits: Do *not* use a fully qualified URL.You must use a relative URI.

Example: /Web pages/logoff.html

More information:

Configure Full Logoff (see page 107)
How to Configure Full Logoff for Single Sign-on (see page 108)
Security Issues Related to Caching HTTP Header Resources (see page 133)

LowerCaseHTTP

Specifies whether the Web Agent uses uppercase or lowercase HTTP headers. Some web servers may be case-sensitive. Set this parameter to no to specify uppercase headers.

Default: Yes

Note: This parameter does *not* apply to IIS agents.

More information:

Use Lower Case HTTP in Headers (for Sun Java System, Apache, Domino) (see page 134)

MasterCookiePath

Specifies the path for the primary-domain session cookies created by the cookie provider. For example, if this parameter is set to /siteminderagent, all session cookies that the cookie provider creates will have the /siteminderagent path. If this parameter is not set in the Cookie Provider Agent, the default value is used.

Default: / (root)

More information:

Specify the Cookie Path for Agent Cookies (see page 111)

MaxResourceCacheSize

Specifies the maximum number of entries that the Web Agent keeps in its resource cache. An entry contains the following information:

- A Policy Server response about whether a resource is protected
- Any additional attributes returned with the response

When the maximum is reached, new resource records replace the oldest resource records.

If you set this value to a high number, be sure that sufficient system memory is available.

If you are viewing Web Agent statistics using the OneView Monitor, you may notice that the value shown for the ResourceCacheCount is greater than the value you specified for the MaxResourceCacheSize parameter. This is not an error. The Web Agent uses the MaxResourceCacheSize parameter as a guideline and the values may at times differ because the MaxResourceCacheSize parameter represents the maximum number of average-sized entries in the resource cache. The actual cache entries are most likely larger or smaller than the pre-determined average size; therefore, the effective maximum number of entries may be more or less than the value specified.

Note: For Web Agents that use shared memory, such as the Agent on Apache 1.x and the framework Agents, the cache is pre-allocated to a constant size based on the MaxResourceCacheSize value and will not grow.

Default: (Domino web servers) 1000

Default: (IIS and Sun Java System web servers) 700

Default: (Apache web servers) 750

Note: If you change the value of this parameter, you must restart the web server to apply the change.

More information:

Set the Maximum Resource Cache Size (see page 183)
Disable the Resource Cache (see page 184)
Parameters Requiring a Server Restart when Changed (see page 25)

MaxSessionCacheSize

Specifies the maximum number of users the Agent maintains in its session cache. The session cache stores the session IDs of users who authenticate successfully. If those users access another resource in the same realm during the same session, the Agent uses the information from the session cache instead of calling the Policy Server. When the maximum number is reached, the Agent replaces the oldest user records with new user records.

Base the value of this parameter on the number of users that you expect to access and use resources for a sustained period of time. If you set this value to a high number, ensure that sufficient system memory is available.

Default: (Domino web servers) 1000

Default: (IIS and Sun Java System web servers) 700

Default: (Apache web servers) 750

Note: If you change the value of this parameter, you must restart the web server to apply the change.

More information:

Set the Maximum User Session Cache Size (see page 184)

MaxTimeoutURL

Specifies the URL where the Web Agent should redirect the user when the maximum time-out for the session occurs.

Example: http://example.mycompany.com/maxtimeoutpage.html

Default: No default

More information:

Redirect a User after a Session Time-out (see page 98)

MaxUrlSize

Specifies the maximum size (in bytes) of a URL that a Web Agent can handle. Because different web servers have different limitations on URL length, check the documentation from your web server vendor before setting this parameter.

Default: 4096 B

More information:

Set a Maximum URL Size (see page 211)

NTCExt

Specifies the MIME type associated with the NTLM credential collector. This collector gathers NT credentials for resources that are protected by the Windows authentication scheme. This scheme applies to resources on IIS web servers that are accessed by the Internet Explorer browser.

You can have multiple extensions in this parameter. If you are using an Agent Configuration Object, select the multi-value option. If you are using a local configuration file, separate each extension with a comma.

Default: .ntc

More information:

How to Configure the NT Challenge/Response Authentication (IIS Only) (see page 147)
Specify an NTLM Credential Collector (see page 152)
Set Up Credential Collectors for IIS and Domino Web Servers (see page 233)

OverlookSessionForMethods

Specifies whether the Web Agent compares the request method of all HTTP requests against the methods listed in this parameter. If a match occurs, the Web Agent does not create or update an SMSESSION cookie. Also, cookie providers (if configured) are not updated for that request.

Default: No default

More information:

Prevent Session Cookie Creation or Updates (see page 90)

OverlookSessionForMethodUri

Specifies whether the Web Agent compares the URI from all HTTP requests against the URI listed in this parameter. If a match occurs, the Web Agent does not create or update an SMSESSION cookie. Also, cookie providers (if configured) are not updated for that request.

Default: No default.

More information:

Prevent Session Cookie Creation or Updates Based on Method and URI (see page 91)

OverlookSessionForUrls

Specifies whether the Web Agent compares the URLs from all HTTP requests against the URLs listed in this parameter. If a match occurs, the Web Agent does not create or update an SMSESSION cookie. Also, cookie providers (if configured) are not updated for that request.

Default: No default

Example: Use a relative URL, such as /MyDocuments/index.html. Do not use an absolute URL (http://fqdn.host/MyDocuments/index.html)

More information:

Prevent Session Cookie Creation or Updates (see page 90)

OverrideIgnoreExtFilter

Specifies a list of strings you want the Web Agent to match against all URIs. This helps you protect resources whose extensions are normally ignored by the Web Agent, or any files or applications that do not have extensions. If the URI matches one of the strings in the list, the Web Agent checks with the Policy Server to determine if the resource is protected.

It is better to specify more general strings instead of exact paths. You can also include a partial string to protect a group of resources. For example, the string /servlet/ protects the following resources:

- /dira/app1/servlet/app
- /dirb/servlet/app1
- /dirc/mydir/servlet/app2

Default: No default

More information:

How to Protect Resources Without Periods or Extensions (see page 215)
Protect Resources Without Extensions (see page 216)

P3PCompactPolicy

Determines whether custom responses comply with the Platform for Privacy Preferences Project (P3P) response headers. P3P compact policies use tokens representing the specific elements from the P3P terminology. If you set the P3PCompactPolicy parameter to the appropriate policy syntax, it ensures that custom responses are set with the correct P3P response header when a P3P compact policy is specified for the Web Agent.

Default: No default

Example: NON DSP COR CURa TAI (these represent: none, disputes, correct, current/always, and tailoring, respectively)

Note: This parameter is *not* supported on Apache 1.3 or Domino Web Agents.

More information:

How to Support a P3P Compact Policy with your SiteMinder Web Agent (see page 73)

PersistentCookies

Specifies whether the agent allows single sign-on for multiple browser sessions. When this is enabled, users who authenticate during one browser session will retain single sign-on capabilities for subsequent browser sessions.

If you set the value of the autoauthorizeoptions parameter to yes, set the value of the PersistentCookies parameter to no.

To enable persistent cookies, you must also set the TransientIDCookies parameter to no.

This parameter affects the following parameters:

- TransientIDCookies
- PersistentIPCheck

Default: No

More information:

How to Configure Single Sign-On (see page 83)
Set Persistent Cookies (see page 85)
Compare IP Addresses to Prevent Security Breaches (see page 227)

PersistentIPCheck

Instructs the Web Agent to compare the IP address from the last request (stored in a persistent cookie) with the IP address in the current request to see if they match. If the IP addresses do not match, the Web Agent rejects the request.

Note: SiteMinder identity cookies are unaffected by IP checking.

This parameter affects the following parameters:

- PersistentCookies

Default: Yes

More information:

Configure IP Address Validation (see page 131)
Compare IP Addresses to Prevent Security Breaches (see page 227)

PostPreservationFile

Enables the transfer of POST preservation data between Traditional and Framework Agents by specifying the path to *one* of the following POST-preservation-template files:

- tr2fw.pptemplate—Indicates that resources hosted on a server running a traditional agent are protected by an FCC running on a Framework agent.
- fw2tr.pptemplate—Indicates that resources hosted on a server running a Framework agent are protected by an FCC running on a Traditional agent.

Default: No default

Example: *web_agent_home*/samples/forms/fw2tr.pptemplate

More information:

Enable Post Preservation between Framework and Traditional Agents (see page 245)

PreserveHeaders

Specifies whether the Web Agent saves the existing HTTP headers instead of replacing them when new headers are created. Set this parameter to yes for Sun Java System, Domino, and Apache Web Agents.

Default: No

More information:

Preserve HTTP Headers (see page 132)

PreservePostData

Specifies whether the Web Agent preserves POST data when redirecting requests. When the user is challenged for advanced authentication, such as forms or certificate authentication, the post data is preserved during the authentication phase.

Default: Yes

More information:

Disable POST Preservation (see page 246)

ProxyAgent

Specifies if a Web Agent is acting as a reverse proxy agent.

When the value of this parameter is yes, the SiteMinder Web Agent on the front-end server preserves the original URL requested by the user in the SM_PROXYREQUEST HTTP header. This header is created whenever protected and unprotected resources are requested. The back-end server can read this header to obtain information about the original URL.

Default: No

Note: This parameter applies to Apache Web Agents only.

More information:

SiteMinder Reverse Proxy Deployment Considerations (see page 198)

ProxyDefinition

Specifies the IP address of a proxy (such as a cache device) that requires the use of a custom HTTP header to resolve requester IP addresses.

Default: No default

Limits: The string must contain an IP address. Do *not* use server names or fully qualified DNS host names.

More information:

Configure IP Address Validation (see page 131)

ProxyTimeout

Specifies the number of seconds the reverse proxy waits for the Web Agent deployed behind it to respond to a request.

Default: No default

Note: This parameter applies to Apache Web Agents only.

More information:

SiteMinder Reverse Proxy Deployment Considerations (see page 198)

ProxyTrust

Instructs the Web Agent for the destination server to trust the authorizations made by the proxy server. This is more efficient because the Web Agent for the destination server does not need to reauthorize users.

Default: No

More information:

Configure Agents that Sit behind Proxy Servers (see page 188)
SiteMinder Reverse Proxy Deployment Considerations (see page 198)

PSPollInterval

Specifies how often (in seconds) the Web Agent contacts the Policy Server to retrieve information about policy changes or dynamically updated keys. Higher numbers (longer intervals) decrease network traffic. Lower numbers (shorter intervals) increase network traffic.

Default: 30

Limit: 1

Example: If the level of traffic in your network is moderate, or your website does not change often, you can increase the interval to reduce network traffic.

More information:

Change How Often an Agent Checks for Policy or Key Updates (see page 52)
Web Agents and Dynamic Key Rollovers (see page 22)

RemoteUserVar

Instructs the Web Agent to populate the REMOTE_USER variable based on the value from an HTTP-WebAgent-Header-Variable response attribute. Use this to integrate with legacy applications. Enter only the name of the response variable.

Example: To return an HTTP-WebAgent-Header-Variable such as "user=aperson", set the RemoteUserVar parameter to user.

Default: No default

More information:

Configure the Web Agent to set the REMOTE_USER Variable (see page 120)

ReqCookieErrorFile

Specifies a customized error page to which users are redirected if a cookie with basic credentials is not returned by the browser when the RequireCookies parameter is set to yes.

Example: http://yourcompany.com/need_cookies.htm

More information:

Custom Error Handling For Applications (see page 139)
How to Set Up Error Handling (see page 141)

RequireCookies

Specifies whether SiteMinder requires cookies. SiteMinder uses cookies to do the following:

- Secure a single sign-on environment
- Track session timeouts
- Track idle time-outs.

Important! If you configure the Web Agent to require cookies, a user's Web browser must accept HTTP cookies. If the browser does not, the user receives an error message from the Agent denying the user access to all protected resources.

Default: Yes

More information:

How to Configure Single Sign-On (see page 83)
Require Cookies for Basic Authentication (see page 84)
Custom Error Handling For Applications (see page 139)

ResourceCacheTimeout

Specifies the number of seconds that resource entries remain in the cache. If a user tries to access a protected resource after the time interval has been exceeded, the Web Agent removes the cached entries and contacts the Policy server.

Default: 600 (10 minutes)

Note: If you change the value of this parameter, you must restart the web server to apply the change.

More information:

Web Agent Cache (see page 181)
Control How Long Resource Enteries Remain Cached (see page 182)
Parameters Requiring a Server Restart when Changed (see page 25)

SaveCredsTimeout

Specifies the number of hours that a persistent cookie containing the user credentials will be saved. During this time interval, the Web Agent authenticates the user with the data stored in the cookie. After this time interval expires, the cookie is removed and the Web Agent challenges the user again.

Default: 720 (30 days)

More information:

Set a Time-out for Saved Credentials (see page 96)

SCCExt

Specifies a MIME type for an SSL Credential Collector.

Default: .scc

More information:

Set Up Credential Collectors for IIS and Domino Web Servers (see page 233)

SecureApps

Prevents the Agent from authorizing URLs from an unauthorized user. If your Web Agent is configured to ignore requests for files ending with certain extensions, an attacker may attempt to access resources by creating a false URL.

For example, if you have a resource with the following URL:

```
/scripts/myapp
```

An attacker may attempt to gain access by creating a false URL like the one in the following example:

```
/scripts/myapp/junk.jpg
```

If the value of the SecureApps parameter is set to no, the request for /scripts/myapp/junk.jpg would be automatically authorized if the Web Agent was set to ignore requests for .jpg files.

If the value of the SecureApps parameter is set to yes, the Web Agent attempts to discover if the resource is legitimate or if the URL is false.

Default: No

More information:

Secure Applications (see page 217)

SecureURLs

Specifies whether the Web Agent encrypts the SiteMinder query parameters in a redirect URL. You can use this setting to provide additional security for requested resources protected by an advanced authentication scheme, Password Services, or when a request invokes the Cookie Provider.

Important! The Web Agent only encrypts data sent between SiteMinder components. The data sent for redirects to non-SiteMinder applications is not encrypted.

The following SiteMinder credential collectors and applications support the SecureUrls functionality:

- HTML Forms authentication
- Cert And Forms authentication
- SSL Authentication
- Cert or Forms authentication
- NTLM authentication
- ACE authentication
- SafeWord authentication
- User self registration
- Multi-domain Single Sign-on with Cookie Provider
- FCC-based Password Services (not CGI- or JSP-based)

Default: No

More information:

Configure SecureUrls with Single Sign-on (see page 106)
Query String Encryption of Redirect URLs (see page 207)
Query String Encryption of Redirect URLs and FCC-based Password Services (see page 208)
Encrypt Query String Parameters in Redirection URLs (see page 209)

ServerErrorFile

Instructs the Web Agent to display a custom error page to users who encounter server errors. Specify a file path or URL for this parameter.

Default: No default

More information:

Custom Error Handling For Applications (see page 139)
How to Set Up Error Handling (see page 141)
Server Error 500 Appears Instead of Custom Error Page (see page 297)

ServerPath

Specifies a unique path to each web server instance when a Web Agent is configured to use multiple instances of a web server. The ServerPath creates a unique identifier for the Web Agent's caching, logging, and health-monitoring resources.

Default: Empty

Example: If there are four web server instances, each loading a Web Agent, then each server's WebAgent.conf file should have the ServerPath parameter set to a unique value. You can set the ServerPath parameter to the directory where the web server's log file is stored, such as *server_instance_root*/logs.

Note: This parameter applies to Apache and Sun Java System agents only.

More information:

WebAgent.conf file for Framework Agents (see page 32)
Manage Web Agents with Multiple Web Server Instances (see page 54)
Set the ServerPath Parameter for Windows Systems (see page 54)
Set the ServerPath Parameter for UNIX Systems (see page 55)
Additional Configurations Requiring the ServerPath Parameter (see page 56)

SessionGracePeriod

Specifies the number of seconds during which a SiteMinder session (SMSESSION) cookie will not be regenerated. Cookies are not regenerated when *all* of the following conditions are met:

- There is no URL SMSESSION cookie.
- The difference between the current time and the last access time of the received SMSESSION cookie is less than or equal to the SessionGracePeriod.
- The amount of time between the current time and the time when the received cookie would have been idle exceeds two grace periods. For example, if your grace period is 25 minutes and the idle time-out is 60 minutes, SiteMinder regenerates a session cookie after 10 minutes, because then there are less than two grace periods (50 minutes) of time left before the session goes idle.

Default: 30

More information:

Modify the Session Grace Period (see page 95)

SessionUpdatePeriod

Specifies how often (in seconds) a Web Agent redirects a request to the Cookie Provider to set a new cookie. Refreshing the master cookie decreases the possibility that it will expire due to an idle time-out of the SiteMinder session.

Default: 60

More information:

Specify the Cookie Provider (see page 86)
Modify the Session Update Period (see page 87)

SetRemoteUser

Specifies a value for the REMOTE_USER variable that some legacy applications may require.

Default: No

More information:

How the IIS Web Agent Populates the REMOTE_USER Variable (see page 119)
Configure the Web Agent to set the REMOTE_USER Variable (see page 120)
Record the User Name and Transaction ID in IIS 6.0 Server Logs (see page 160)

SFCCExt

Specifies the MIME type for the SSL Forms Credential Collector.

Default: .sfcc

More information:

Set Up Credential Collectors for IIS and Domino Web Servers (see page 233)

SkipDominoAuth

Directs the SiteMinder Web Agent to authenticate users instead of using the Domino authentication mechanism. You should also set this parameter to yes when users are *not* stored in the Domino directory.

This parameter affects the following parameters:

- DominoSuperUser

Default: Yes

Note: This parameter applies to Domino Web Agents only.

More information:

Authenticate Users with the Domino Server (see page 262)
Force SiteMinder to Authenticate Users (see page 264)
Authenticate as the Domino Super User (see page 265)
Authenticate as the Actual User or the Default User (see page 265)
Considerations for Creating Policies on Domino Servers (see page 278)

SSOTrustedZone

Defines an ordered (case-sensitive) list of trusted SSOZoneNames of trust for a single sign-on security zone. Use SM to add the default zone if necessary. Agents always trust their own SSOZoneName above all other trusted single sign-on zones.

Default: Empty (SM or the SSOZoneName if provided)

Limits: Multi-valued

More information:

The Order of Trust and Failover (see page 293)

SSOZoneName

Specifies the (case-sensitive) name of the single sign-on security zone a Web Agent supports. The value of this parameter is prepended to the name of the cookie a Web Agent creates. This helps you associate cookies with their respective cookie domains. When this parameter is not empty, SiteMinder generates cookies using the following convention:

ZonenameCookiename.

Default: Empty (uses SM as a zone name, which gives the cookies the following default names):

- SMSESSION
- SMIDENTITY
- SMDATA
- SMTRYNO
- SMCHALLENGE
- SMONDENIEDREDIR

Limits: Single-valued

Example: Setting the value to Z1 creates the following cookies:

- Z1SESSION
- Z1IDENTITY
- Z1DATA
- Z1TRYNO
- Z1CHALLENGE
- Z1ONDENIEDREDIR

More information:

Trusted Zone Order (see page 285)
Configure Security Zones (see page 290)
Specify the Single Sign-on Zone for the Agent (see page 292)

StoreSessioninServer

Specifies whether single-use session cookies are used. When the value of the StoreSessioninServer parameter is yes, a single-use session cookie is created and stored on the session server. Cookie providers and Web Agents access the cookie from the session server.

Cookie providers and Web Agents replace the session cookie in a URL with a GUID that corresponds to the single-use session cookie stored on the session server.

When the value of the StoreSessioninServer parameter is no, the session cookie is passed directly in the URL

Default: No

SuppressServerHeader

Prevents an IIS Web Agent from returning the Server HTTP Header in its responses. When the value of this parameter is set to no, the Web Agent sends the Server header with its responses and the IIS Web server passes it along to the client. When the value of this parameter is set to yes, the web agent does not send the Server header in its responses.

Default: No

Note: This parameter applies to IIS Web Agents only.

More information:

Remove the Server HTTP Header if Using the URLScan Utility (see page 60)

TargetAsRelativeURI

Instructs the Web Agent to use a relative URI instead of a fully qualified URL when directing requests to a credential collector and target resource. Using a relative URI prevents requests from being processed by credential collectors on other systems installed with Web Agents. Enabling this parameter also causes the Web Agent to reject any target that does not begin with a forward slash (/).

Note: This setting applies to all credential collectors *except* the cookie credential collector (CCC). The CCC must use a fully-qualified domain name for this parameter. OnAuthAccept responses will not work properly with a CCC if a relative URI is used.

Default: No

More information:

Use a Relative Target for Credential Collector Redirects (see page 242)

TraceAppend

Adds new logging information to the end of an existing log file instead of rewriting the entire file each time logging is invoked.

Default: No

More information:

Configure Trace Logging (see page 168)

TraceConfigFile

Specifies the location of the WebAgentTrace.conf configuration file that determines which components and events to monitor.

Default: No default

Example: C:\Program Files\ca\webagent\config\WebAgentTrace.conf

More information:

Configure Trace Logging (see page 168)

TraceDelimiter

Specifies the custom character that separates the fields in the trace.conf file.

Default: No default

Example: |

More information:

Configure Trace Logging (see page 168)

TraceFile

Enables trace logging.

Default: No

More information:

Configure Trace Logging (see page 168)

TransientIPCheck

Instructs the Web Agent to compare the IP address from the last request (stored in a transient cookie) with the IP address in the current request to see if they match. If the IP addresses do not match, the Web Agent rejects the request.

Note: SiteMinder identity cookies are unaffected by IP checking.

This parameter affects the following parameters:

- TransientIDCookies

Default: No

More information:

Compare IP Addresses to Prevent Security Breaches (see page 227)

TraceFileName

Specifies the full path to the trace log file.

Default: No default
Example: C:\Program Files\ca\webagent\log\trace.log

More information:

Configure Trace Logging (see page 168)

TraceFileSize

Specifies (in megabytes) the maximum size of a trace file. The Web Agent creates a new file when this limit is reached.

Note: This feature is not supported for Apache 1.x and Sun Java System on UNIX systems. Use the default or leave this setting blank.

Default: 0 (a new log file is not created)

Example: 20 (MB)

More information:

Configure Trace Logging (see page 168)

TraceFilesToKeep

Specifies the number of Web Agent trace log files that are kept. New trace logs are created in the following situations:

- When the Web Agent starts.
- When the size limit of the trace log (specified by the value of the TraceFileSize parameter) is reached.

Changing the value of this parameter does *not* automatically delete any existing trace logs which exceed the number that you want to keep. For example, If your system has 500 trace logs stored, and you decide to keep only 50 of those files, the Web Agent does *not* delete the other 450 trace logs.

Setting the value of this parameter to zero retains all the trace logs.

Default: 0

More Information:

Limit the Number of Trace Log Files Saved (see page 177)

TraceFormat

Specifies how the trace.conf file displays the messages. Choose *one* of the following options:

- default—uses square brackets [] to enclose the fields.
- fixed—uses fields with a fixed width.
- delim—uses a character of your choice to delimit the fields.
- xml—uses XML-like tags. A DTD or style sheet is *not* provided with the Web Agent.

Default: default (square brackets)

More information:

Configure Trace Logging (see page 168)

TrackSessionDomain

Instructs the Web Agent to encrypt and store the intended domain of a session cookie within the session cookie itself. When the session cookie is presented for subsequent requests, The Web Agent compares the intended domain stored within the session cookie against the domain of the requested resource. If the domains do *not* match, the Web Agent rejects the request.

For example, when the value of this parameter is set to yes, session cookies intended for use with operations.example.com would be rejected by the Web Agent if they were presented at finance.example.com.

Default: No

More information:

Validate a Session Cookie Domain (see page 89)

TransientIDCookies

Specifies whether the Agent Identity (SMIDENTITY) cookie is transient or persistent. Use persistent cookies to give users single sign-on capability across multiple browser sessions. As long as the SiteMinder session has not expired, users will not have to reauthenticate.

Use transient cookies if you want users to reauthenticate to a single sign-on environment for each separate browser session.

This parameter affects the following parameters:

- TransientIPCheck
- PersistentIPCheck

Default: No

More information:

Control Identity Cookies (see page 94)
Set Persistent Cookies (see page 85)

UseAnonAccess

Instructs the IIS Web Agent to execute the web application as an anonymous user, instead of using credentials of the proxy user.

Default: No

Note: This parameter applies to IIS Web Agents only.

More information:

How the IIS Web Agent Populates the REMOTE_USER Variable (see page 119)
Enable Anonymous User Access (see page 145)

UseHTTPOnlyCookies

Instructs the Web Agent to set the HTTP-only attribute on the cookies it creates. When a Web Agent returns a cookie with this attribute to a user's browser, the contents of the cookie cannot be read by a script, even a script from the web site which originally set the cookie. This helps prevent any sensitive information in the cookie from being sent to an unauthorized third party through a script.

Default: No

More information:

Safeguard Information in Cookies with HTTP-Only Attribute (see page 227)

UseNetBIOSforIISAuth

Specifies whether the IIS 6.0 Web Agent sends the user principal name (UPN) or the NetBIOS name to the IIS 6.0 web server for IIS user authentication.

Note: This parameter is valid only if an Active Directory user store is associated with the Policy Server.

If you enable this parameter, the Policy Server extracts the UserDN, the UPN, and the NetBIOS name from the Active Directory during SiteMinder authentication, and sends this data back to the IIS 6.0 Web Agent.

Depending on whether or not you selected the Use Authenticated User's Security Context option for the user directory with the Administrative UI and how you set the UseNetBIOSforIIAuth parameter, a user's logon credentials are sent as follows:

- When the UseNetBIOSforIISAuth parameter is set to no, the IIS 6.0 Web Agent sends the UPN name.
- When the UseNetBIOSforIISAuth parameter is set to yes, the Web Agent sends the NetBIOS name.

The IIS web server authenticates the user with the credentials it receives from the Web Agent.

Default: No

Note: This parameter applies to IIS Web Agents only.

More information:

Use the NetBIOS Name or UPN for IIS Authentication (see page 146)

UseSecureCookies

Sends cookies to web servers using secure (HTTPS) connections. Enable this parameter to increase security between browsers and web servers.

When this setting is enabled, users in single sign-on environments who move from an SSL web server to a non-SSL web server will have to reauthenticate. Secure cookies cannot be passed over traditional HTTP connections.

Default: No

More information:

Set Secure Cookies (see page 92)

UseServerRequestIp

Instructs the Web Agent to resolve the AgentName according to the physical IP address of a virtual web server. Use this parameter to increase security if a web server uses IP addresses for virtual server mappings. If this parameter is set to no, the Web Agent resolves the AgentName according to the host name in the HTTP Host header of the client's request.

For Domino servers, this parameter is supported only for Domino 6.x. If this parameter is enabled for an Agent on other Domino versions, the Web Agent uses the default Agent name.

For IIS Web Agents configured for SSL communication and virtual hosts, you must set this parameter to yes. IIS does not allow virtual host mappings using host names with SSL enabled.

Default: No

More information:

Resolve Agent Identity by IP Address (see page 72)

ValidTargetDomain

Specifies the domains to which a credential collector is allowed to redirect users. If the domain in the URL does not match the domains set in this parameter, the redirect is denied.

Default: No default

More information:

Define Valid Target Domains for CCC Processing (see page 243)

Agent Configuration Parameters Used Only for Apache Servers

The following list shows the configuration parameters used only for the Apache Web Agent in alphabetical order:

DeleteCerts

Specifies if the certificates stored on a Stronghold server will be removed when the Web Agent finishes using them.

Default: No

More information:

Delete Certificates from Stronghold (Apache Agent Only) (see page 280)

GetPortFromHeaders

Directs the Web Agent to obtain the port number from the HTTP HOST request header instead of obtaining it from the web server service structures.

Default: No

More information:

Use the HTTP HOST Request for the Port Number (see page 61)

HttpsPorts

Specifies the secure ports the Web Agent listens on if you are using an SSL connection to the web server. If you specify a value for this parameter, you must include all the ports for all the web servers that serve secure requests. If you do not specify a value, the Web Agent reads the HTTP scheme from the web server's context.

If a server is behind an HTTPS accelerator (which converts HTTPS to HTTP), the requests are treated as SSL connections by your browser.

Default: Empty

Example: 80

Example: (multiple ports) 80,8080,8083

More information:

Define HTTPS Ports (see page 137)
SiteMinder Reverse Proxy Deployment Considerations (see page 198)

LegacyTransferEncodingBehavior

Specifies the type of message encoding used by the Web Agent. When the value of this parameter is set to no, transfer-encoding is supported.

When the value of this parameter is set to yes, content encoding is used. The transfer-encoding header is ignored and only the content-length header is supported.

Default: No

ProxyAgent

Specifies if a Web Agent is acting as a reverse proxy agent.

When the value of this parameter is yes, the SiteMinder Web Agent on the front-end server preserves the original URL requested by the user in the SM_PROXYREQUEST HTTP header. This header is created whenever protected and unprotected resources are requested. The back-end server can read this header to obtain information about the original URL.

Default: No

More information:

SiteMinder Reverse Proxy Deployment Considerations (see page 198)

ProxyTimeout

Specifies the number of seconds the reverse proxy waits for the Web Agent deployed behind it to respond to a request.

Default: No default

More information:

SiteMinder Reverse Proxy Deployment Considerations (see page 198)

ServerPath

Specifies a unique path to each web server instance when a Web Agent is configured to use multiple instances of a web server. The ServerPath creates a unique identifier for the Web Agent's caching, logging, and health-monitoring resources.

Default: Empty

Example: If there are four web server instances, each loading a Web Agent, then each server's WebAgent.conf file should have the ServerPath parameter set to a unique value. You can set the ServerPath parameter to the directory where the web server's log file is stored, such as *server_instance_root*/logs.

More information:

WebAgent.conf file for Framework Agents (see page 32)
Manage Web Agents with Multiple Web Server Instances (see page 54)
Set the ServerPath Parameter for Windows Systems (see page 54)
Set the ServerPath Parameter for UNIX Systems (see page 55)
Additional Configurations Requiring the ServerPath Parameter (see page 56)

Agent Configuration Parameters Used Only for Domino Servers

The following list shows the configuration parameters used only for the Domino Web Agent in alphabetical order:

DominoDefaultUser

Specifies the name by which the Domino Web Agent identifies the users that SiteMinder has previously authenticated against another directory to the Domino server.

Important! This parameter must be encrypted if it is stored in a local configuration file. Use the encryptkey tool to encrypt this parameter. Do not change it by editing the local configuration file directly.

Default: No default

More information:

Authenticate Users with the Domino Server (see page 262)
Authenticate as the Actual User or the Default User (see page 265)
Modify the Domino Default User and the Domino Super User (see page 266)
Use Encryptkey to Set the Domino Default or Super User (see page 267)

DominoLegacyDocumentSupport

Specifies how a Web Agent handles user requests for protected Lotus Notes documents in a Domino environment. Setting this parameter to yes grants users ReadForm permission only for the requested document.

Default: No

More information:

Control Access to Lotus Notes Documents (see page 271)

DominoLookUpHeaderForLogin

Instructs the Domino Web Agent to ask the Domino web Server if the user requesting access to a resource is unique or ambiguous within the Domino user directory. This helps if a user requesting access to a resource has the same name as other users in the user directory.

Default: No

More information:

Use a SiteMinder Header for Authentication (see page 268)

DominoMapUrlForRedirect

Instructs the Web Agent to map (normalize) the URL from the Domino server representation to a URL-friendly name for the redirect to the Forms Credential Collector (FCC). The FCC can process the request for the requested Domino resource. If this parameter is missing, the default behavior occurs. If this parameter is set to no, the Web Agent does not map the URL, and performs FCC redirects using the original Domino server representation.

The DominoNormalizeUrls parameter must also be set to yes, otherwise the URL will not be normalized.

Default: Yes

More information:

Map URLs for FCC Redirects (see page 267)

DominoNormalizeUrls

Specifies if the SiteMinder Web Agent converts Domino URLs to a URL-friendly name before redirecting them to a Forms Credential Collector.

The MapUrlsForRedirect parameter must also be set to yes for the Domino URLs to be converted.

If the DominoNormalizeUrls parameter is set to no, URLs will *not* be normalized, even if the MapUrlsForRedirect parameter is set to yes.

Important! If you set the DominoNormalizeUrls parameter to no, you cannot protect individual documents within a Notes database; you can only protect the entire database or subdirectories of the Domino Web server.

Default: Yes

More information:

Map URLs for FCC Redirects with a Domino Web Agent (see page 269)
Disable URL Normailization (see page 270)

DominoSuperUser

Identifies a user who has access to all resources on the Domino server, and ensures that all users successfully logged into SiteMinder will be logged into Domino as the Domino SuperUser.

This value can be encrypted.

This parameter affects the following parameters:

- SkipDominoAuth

Default: No default

More information:

Authenticate Users with the Domino Server (see page 262)
Authenticate as the Domino Super User (see page 265)
Modify the Domino Default User and the Domino Super User (see page 266)
Use Encryptkey to Set the Domino Default or Super User (see page 267)

DominoUseHeaderForLogin

Instructs the Domino Web Agent to pass the SiteMinder header value to the Domino web server. The Domino server uses the header data to identify a user in its user directory.

Default: No default

More information:

Use a SiteMinder Header for Authentication (see page 268)

SkipDominoAuth

Directs the SiteMinder Web Agent to authenticate users instead of using the Domino authentication mechanism. You should also set this parameter to yes when users are *not* stored in the Domino directory.

This parameter affects the following parameters:

- DominoSuperUser

Default: Yes

More information:

Authenticate Users with the Domino Server (see page 262)
Force SiteMinder to Authenticate Users (see page 264)
Authenticate as the Domino Super User (see page 265)
Authenticate as the Actual User or the Default User (see page 265)
Considerations for Creating Policies on Domino Servers (see page 278)

Agent Configuration Parameters Used Only for IIS Servers

The following list shows the configuration parameters used only for the IIS Web Agent in alphabetical order:

AppendIISServerLog

Instructs the Web Agent to add the authenticated user name and SiteMinder transaction ID to the IIS server log on a separate line.

Default: No

Note: This parameter applies to IIS 6.0 Web Agents only.

More information:

Record the User Name and Transaction ID in IIS 6.0 Server Logs (see page 160)

DefaultPassword

Specifies a default password for the associated Windows user that is used to access IIS resources as a proxy user.

Important! If you want to encrypt this parameter, set it centrally in the Agent Configuration Object. If this parameter is set in a local configuration file, it will not be encrypted and will be less secure.

Default: No default

More information:

Use an IIS Proxy User Account (IIS Only) (see page 144)
How the IIS Web Agent Populates the REMOTE_USER Variable (see page 119)

DefaultUsername

Specifies the name of a Windows user that is used to access IIS resources as a proxy user. When users want to access resources on an IIS web server protected by SiteMinder, they may not have the necessary server access privileges. For example, if users are stored in an LDAP user directory on a UNIX system, those users may not have access to the Windows system with the IIS web server.

The Web Agent must use this NT user account, which is assigned by an NT administrator, to act as a proxy user account for users granted access by SiteMinder.

Default: No default

More information:

Use an IIS Proxy User Account (IIS Only) (see page 144)
How the IIS Web Agent Populates the REMOTE_USER Variable (see page 119)

DisablePostDataLimit

Specifies whether a Web Agent observes the 64 KB data-size limit when preserving or filtering POST data. This does not affect the standard POST operation, but it does affect the following:

- POST preservation
- eTelligent Rules Post variables
- TransactionMinder authentication schemes

Default: No (limit enforced)

Important! Change this parameter to yes at your own risk.

Note: This parameter applies to IIS 5.0 Web Agents only.

More information:

Limit Size of Post Data (IIS 5.0 Agents only) (see page 58)

ForceIISProxyUser

Specifies whether the Web Agent uses an IIS proxy account to grant access to requested resources on IIS web servers to users who normally lack sufficient privileges to access the IIS web server.

This parameter affects the following parameters:

- DefaultUserName
- DefaultPassword

Default: No

More information:

How the IIS Web Agent Populates the REMOTE_USER Variable (see page 119)
Use an IIS Proxy User Account (IIS Only) (see page 144)

InsecureServer

Allows Web Agents for IIS 5 to enforce native IIS security mechanisms by providing a Windows user security context. Add this parameter to the Agent Configuration Object or local configuration file with the value you want.

If this parameter is set to yes, the Web Agent stores encrypted credentials in paged memory, which can be written to the operating system's page file and saved to a hard disk.

Important! If your hard disk is stolen or compromised, confidential data could be exposed.

If this parameter is no, the Web Agent stores encrypted credentials in protected kernel memory. This setting is more secure, but it places more demands on the physical memory of your IIS server.

Default: No

Note: This parameter applies to IIS 5.0 Web Agents only. It is not used in Framework Web Agents.

More information:

Store Encrypted Credentials in a Page File (IIS 5.0 Only) (see page 143)
Parameters Not Used by Framework Agents (see page 24)

SuppressServerHeader

Prevents an IIS Web Agent from returning the Server HTTP Header in its responses. When the value of this parameter is set to no, the Web Agent sends the Server header with its responses and the IIS Web server passes it along to the client. When the value of this parameter is set to yes, the web agent does not send the Server header in its responses.

Default: No

More information:

Remove the Server HTTP Header if Using the URLScan Utility (see page 60)

UseAnonAccess

Instructs the IIS Web Agent to execute the web application as an anonymous user, instead of using credentials of the proxy user.

Default: No

More information:

How the IIS Web Agent Populates the REMOTE_USER Variable (see page 119)
Enable Anonymous User Access (see page 145)

UseNetBIOSforIISAuth

Specifies whether the IIS 6.0 Web Agent sends the user principal name (UPN) or the NetBIOS name to the IIS 6.0 web server for IIS user authentication.

Note: This parameter is valid only if an Active Directory user store is associated with the Policy Server.

If you enable this parameter, the Policy Server extracts the UserDN, the UPN, and the NetBIOS name from the Active Directory during SiteMinder authentication, and sends this data back to the IIS 6.0 Web Agent.

Depending on whether or not you selected the Use Authenticated User's Security Context option for the user directory with the Administrative UI and how you set the UseNetBIOSforIIAuth parameter, a user's logon credentials are sent as follows:

- When the UseNetBIOSforIISAuth parameter is set to no, the IIS 6.0 Web Agent sends the UPN name.
- When the UseNetBIOSforIISAuth parameter is set to yes, the Web Agent sends the NetBIOS name.

The IIS web server authenticates the user with the credentials it receives from the Web Agent.

Default: No

More information:

Use the NetBIOS Name or UPN for IIS Authentication (see page 146)

Agent Configuration Parameters Used Only for Sun Java System Servers

The following list shows the configuration parameters used only for the Sun Java System Web Agent in alphabetical order:

DisableDirectoryList

Specifies whether the Web Agent allows a user to view or browse the contents of a directory without challenging them first. This occurs when *all* of the following conditions are true:

- The realm is set to protect a root resource (/)
- The default web page in the directory (such as index.html) is renamed or deleted.

Default: No

More information:

Restrict Directory Browsing on a Sun Java System Server (see page 63)

EnableOtherAuthTrans

Allows the use of other AuthTrans functions along with SiteMinder.

Default: No

More information:

Handle Multiple AuthTrans Functions (Sun Java System only) (see page 138)

ServerPath

Specifies a unique path to each web server instance when a Web Agent is configured to use multiple instances of a web server. The ServerPath creates a unique identifier for the Web Agent's caching, logging, and health-monitoring resources.

Default: Empty

Example: If there are four web server instances, each loading a Web Agent, then each server's WebAgent.conf file should have the ServerPath parameter set to a unique value. You can set the ServerPath parameter to the directory where the web server's log file is stored, such as *server_instance_root*/logs.

More information:

WebAgent.conf file for Framework Agents (see page 32)
Manage Web Agents with Multiple Web Server Instances (see page 54)
Set the ServerPath Parameter for Windows Systems (see page 54)
Set the ServerPath Parameter for UNIX Systems (see page 55)
Additional Configurations Requiring the ServerPath Parameter (see page 56)

Appendix C: Error Codes

When the Web Agent encounters problems, it generates error codes, which help you diagnose problems with SiteMinder operations. For example, if the Web Agent could not reach the SiteMinder authentication server, the Web Agent would display the code: 20-0002.

The Web Agent can display error codes in the following ways:

- In your Web browser
- As part of a custom server error page

If you configure custom error handling, the Web Agent passes the error codes to a specified URL or customized HTML file that displays the custom error page.

This section contains the following topics:

Miscellaneous 500 HTTP Server Error Codes

This section lists the miscellaneous server error codes.

00-0001

Reason:

Unable to resolve the agent name from an IP address

Action:

Check the agent configuration and ensure that each HOST address served by the web server has a corresponding AgentName mapped to it or that DefaultAgentName is set properly.

00-0002

Reason:

Illegal Characters exist in a URL or characters defined in the BadUrlChars parameter have been detected in a URL.

Action:

Do one of the following:

- Remove the offending characters from the URL
- Remove the characters from the list in the BadUrlChars parameter so they will no longer be blocked.

00-0004

Reason:

An SSLCRED cookie contains a status of error.

Action:

Investigate the Web Agent acting as the secure credential collector (SCC) and verify its configuration.

Typically, this error occurs only when the SCC agent cannot acquire credentials from its environment. This indicates a possible configuration error.

00-0005

Reason:

A FORMCRED cookie contains a status of error.

Action:

Investigate the Web Agent acting as the forms credential collector (FCC) and verify its configuration.

Typically, this error occurs only when the FCC agent cannot acquire credentials from its environment. This indicates a possible configuration error.

00-0006

Reason:

An NTLM Protected Resource was not found in the resource cache as expected.

Action:

Investigate the Windows authentication scheme setup to verify the configuration.

00-0007

Reason:

An ASCII encoding error exists. This is an internal Web Agent error.

Action:

Investigate the web server and Web Agent to diagnose possible service instability.

Contact Customer Support with the Web Agent log and configuration files available for review.

More information:

Contact CA (see page iii)

00-0008

Reason:

SSL Authentication failed. This error indicates a bad certificate or that the user is not authenticated.

Action:

Try a different certificate or investigate the SSL authentication scheme configuration for possible issues.

00-0009

Reason:

Bad or Missing SSL credentials.

Action:

Try a different certificate or username and password pair. Investigate the SSL authentication scheme configuration for possible issues.

00-0010

Reason:

Access Denied. This error indicates a general failure that resulted in blocked access.

Action:

Investigate the Web Agent and Policy Server logs to determine the root cause of the failure.

00-0011

Reason:

Credential Collector Error.This indicates that a general failure in Forms or SSL-based advanced authentication resulted in blocked access.

Action:

Do the following:

- Check the Web Agent and Policy Server logs to determine the root cause of the failure.
- Investigate the advanced authentication scheme setup for issues.

00-0012

Reason:

Encryption Error. This indicates an internal Web Agent error.

Action:

Do the following:

- Investigate the web server and Web Agent to diagnose a possible service instability.
- Review Key Store setup to verify that proper Agent Keys are in use.
- Contact Customer Support and send the Web Agent log and configuration files for review.

More information:

Contact CA (see page iii)

00-0013

Reason:

Agent Configuration Error. One or more errors occurred during startup preventing valid configuration of the Web Agent.

Action:

Do the following:

- On Windows, check the Application Event Log for more information.
- For Apache agents, check the Apache error log for more information.
- For Sun Java System UNIX agents, start Sun Java System from a shell prompt and look for possible errors displayed there through STDERR.
- Check that SmHost.conf file exists (host is registered properly) and contains valid entries.
- Check that Agent Configuration file contains a valid HostConfigFile entry that points to a valid SmHost.conf file.
- Check that AgentConfigObject contains a valid value.

00-0014

Reason:

Could not Log the user out.

Action:

Check the following files for more information:

- Web Agent log file
- Web Agent trace file
- Policy Server log file
- Policy Server trace file

00-0015

Reason:

SiteMinder accounting server answered SM_AGENTAPI_NO to auditing request.

Action:

Check the following files for more information:

- Policy Server log file
- Policy Server trace file

00-0016

Reason:

Unable to resolve the FQ host name.

Action:

Check the Web Agent logs to determine the host name that the Agent is trying to resolve. If the host name is correct, check the DNS settings of the web server on which the agent runs.

00-0017

Reason:

Invalid redirect target found.

Action:

Examine the log file of the Web Agent which is reporting this message to locate the URL being processed (usually an FCC or other advanced authentication URL) and determine if the value of the TARGET CGI parameter appears valid.

HTTP Header-Parsing Error Codes

This section lists the error codes which relate to the parsing of HTTP headers.

10-0001

Reason:

Unable to read the 'SERVER_NAME' HTTP variable.

Action:

Check that the web browser and web server are HTTP 1.0-compliant.

10-0002

Reason:

Unable to read the 'URL' HTTP variable.

Action:

Check that the web browser and web server are HTTP 1.0-compliant.

10-0003

Reason:

Unable to read the 'method' HTTP variable.

Action:

Check that the web browser and web server are HTTP 1.0-compliant.

10-0004

Reason:

Unable to read the 'host' HTTP variable.

Action:

Check that the web browser and web server are HTTP 1.0-compliant.

10-0005

Reason:

Unable to read the 'URI' HTTP variable.

Action:

Check that the web browser and web server are HTTP 1.0-compliant.

10-0007

Reason:

The URL is too long.

Action:

Increase setting of the MaxUrlSize parameter; the default setting is 4096 bytes.

More information:

Set a Maximum URL Size (see page 211)

SiteMinder Communication Error Codes

This section lists the error codes related to communication errors.

20-0001

Reason:

Unable to reach SiteMinder accounting server or an unexpected Policy Server error occurred.

Action:

Do the following:

- Check the Policy Server logs for more detailed information on the error.
- Check connectivity between the Web Agent and the Policy Server by pinging the Policy Server. If a firewall is configured between the Agent and the Policy Server, check that it is not blocking the following service ports:
  - 44441 (accounting)
  - 44442 (authentication)
  - 44443 (authorization)

20-0002

Reason:

Unable to reach SiteMinder authentication server or an unexpected Policy Server error occurred.

Action:

Do the following:

- Check Policy Server logs for more detailed information on the error.
- Check connectivity between the Web Agent and the Policy Server by pinging the Policy Server. If a firewall is configured between the Agent and the Policy Server, check that it is not blocking the following service ports:
  - 44441 (accounting)
  - 44442 (authentication)
  - 44443 (authorization)

20-0003

Reason:

Unable to reach SiteMinder authorization server or an unexpected Policy Server error occurred.

Action:

Do the following:

- Check Policy Server logs for more detailed information on the error.
- Check connectivity between the Web Agent and the Policy Server by pinging the Policy Server. If a firewall is configured between the Agent and the Policy Server, check that it is not blocking the following service ports:
  - 44441 (accounting)
  - 44442 (authentication)
  - 44443 (authorization)

SiteMinder Password Services Error Codes

This section lists the error codes related to password services.

30-0026

Reason:

The Password Services Redirect URL is not available.

Action:

Check that you have configured the redirection URL for password services.

Index

B

C

D

E

F

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W