This documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the “Documentation”) is for your informational purposes only and is subject to change or withdrawal by CA at any time.

This Documentation may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA. This Documentation is confidential and proprietary information of CA and may not be disclosed by you or used for any purpose other than as may be permitted in (i) a separate agreement between you and CA governing your use of the CA software to which the Documentation relates; or (ii) a separate confidentiality agreement between you and CA.

Notwithstanding the foregoing, if you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION “AS IS” WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with “Restricted Rights.” Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2011 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.
Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information you need for your Home Office, Small Business, and Enterprise CA Technologies 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 Technologies product documentation, you can send a message to techpubs@ca.com.

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

## Chapter 1: SiteMinder Secure Proxy Server Overview

- Introduction to the Secure Proxy Server .......................................................... 11
- Proxy Server Architecture .................................................................................. 11
- Traditional Reverse Proxy Server Architecture ................................................. 12
- Secure Proxy Server Architecture ..................................................................... 12
- SPS Components ............................................................................................... 14
- Secure Proxy Server Product Features ............................................................... 16
- Secure Proxy Server Product Limitations .......................................................... 17
- SPS in an Enterprise .......................................................................................... 18
- SPS as a Centralized Access Control Filter ....................................................... 19
- SPS Support for Cookieless Sessions .................................................................. 20
- SPS Support for Extranet Access Control ......................................................... 22

## Chapter 2: Installing and Uninstalling the Secure Proxy Server

- Before You Install .............................................................................................. 25
- Requested Host Registration Information ......................................................... 26
- Requested Apache Web Server Information ...................................................... 27
- Install the SPS on Windows ................................................................................. 27
- Install the SPS on UNIX .................................................................................... 28
- Cancel the SPS Installation ................................................................................ 28
- Run the Configuration Wizard .......................................................................... 29
- Start and Stop the SPS ...................................................................................... 30
- Start the SPS in Single or Multi Process Mode .................................................. 31
- Confirm that the Secure Proxy Server is Functioning ...................................... 32
- Reinstall the SPS ............................................................................................... 32
- Silent Installation and Configuration .................................................................. 33
- Uninstall the SPS from UNIX .......................................................................... 33
- Uninstall the SPS from Windows ..................................................................... 34
- Additional Configuration on the SPS ................................................................. 34
- Modify the Default Location of the SiteMinder Forms ...................................... 35

## Chapter 3: Upgrading the Secure Proxy Server

- SPS Upgrade Overview .................................................................................... 37
- Upgrading from SPS v6.0 .................................................................................. 37
- Additional Tasks for Upgrades .......................................................................... 38
  - Duplicate Custom Settings ............................................................................ 38
Chapter 4: FIPS-140 Support

FIPS Support Overview .................................................. 41
Configuration Process for FIPS ONLY Mode ...................... 42
Migration to FIPS MIGRATE Mode .................................. 42
Migration to FIPS ONLY Mode ...................................... 43

Chapter 5: Using the SPS with Federation Security Services

Federation Security Services Introduction .......................... 45
SPS Use Cases in a SiteMinder Federated Environment .......... 45
Use Case 1: Single Sign-on Based on Account Linking .......... 46
Use Case 2: Single Sign-on Based on User Attribute Profiles .. 47
Use Case 3: Single Sign-on with No Local User Account ....... 48
Use Case 4: Extended Networks ...................................... 49
SPS Roles in a SiteMinder Federated Environment ............... 50
Solutions for SPS Use Cases ........................................... 50
Solution 1: SSO Based on Account Linking ....................... 50
Solution 2: SSO Using User Attribute Profiles .................. 53
Solution 3: SSO with No Local User Account ................... 55
Solution 4: SSO in an Extended Network ......................... 57
Cookieless Federation ................................................ 59
Enable Cookieless Federation at the Consuming Side .......... 61
SPS as a Web Agent Replacement .................................. 61
  Prerequisites for Using the SPS as a Web Agent Replacement 62
  Configuring the SPS as a Web Agent Replacement for Federation 63
SPS as a Federation Gateway .......................................... 63
  Prerequisites for Using the Federation Gateway ............... 65
  Configuring the SPS Federation Gateway ...................... 65
  Limitations of the SPS Federation Gateway .................... 66

Chapter 6: Security Zones on SPS

Overview Single Sign-on Security Zones ......................... 67
Parameters for Security Zones ....................................... 67
Configure SPS Security Zones ...................................... 69

Chapter 7: Configuring the Apache Web Server

Apache Web Server Configuration File ............................. 71
# Chapter 8: Configuring the SPS Server Settings

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPS server.conf File Overview</td>
<td>73</td>
</tr>
<tr>
<td>Modifying the server.conf File</td>
<td>74</td>
</tr>
<tr>
<td>General Server Settings in the server.conf File</td>
<td>74</td>
</tr>
<tr>
<td>HTTP Connection Parameters</td>
<td>74</td>
</tr>
<tr>
<td>Tomcat Tuning Parameters in the server.conf File</td>
<td>75</td>
</tr>
<tr>
<td>Resolve Differences in Cookie Specification for Different Version of Tomcat</td>
<td>76</td>
</tr>
<tr>
<td>Parsing the Equal Sign in a Cookie</td>
<td>77</td>
</tr>
<tr>
<td>Federation Settings in the server.conf File</td>
<td>77</td>
</tr>
<tr>
<td>Logging Settings in the server.conf File</td>
<td>78</td>
</tr>
<tr>
<td>HttpClient Logging</td>
<td>80</td>
</tr>
<tr>
<td>SSL Settings in the server.conf File</td>
<td>82</td>
</tr>
<tr>
<td>Setting for Special Characters within the Cookie</td>
<td>84</td>
</tr>
<tr>
<td>Select Character Set for Code Headers</td>
<td>84</td>
</tr>
<tr>
<td>Caching POST Data</td>
<td>85</td>
</tr>
<tr>
<td>Session Store Settings in the server.conf File</td>
<td>85</td>
</tr>
<tr>
<td>Service Dispatcher Settings in the server.conf File</td>
<td>86</td>
</tr>
<tr>
<td>Proxy and Redirect Settings in the server.conf File</td>
<td>87</td>
</tr>
<tr>
<td>Proxy Service Configuration</td>
<td>87</td>
</tr>
<tr>
<td>Connection Pooling Recommendations</td>
<td>91</td>
</tr>
<tr>
<td>Redirect Service Configuration</td>
<td>92</td>
</tr>
<tr>
<td>Session Scheme Settings in the server.conf File</td>
<td>93</td>
</tr>
<tr>
<td>Establishing a User Session</td>
<td>94</td>
</tr>
<tr>
<td>Default Session Scheme</td>
<td>94</td>
</tr>
<tr>
<td>SSL ID Session Scheme</td>
<td>96</td>
</tr>
<tr>
<td>IP Address Session Scheme</td>
<td>97</td>
</tr>
<tr>
<td>Mini-cookies Session Scheme</td>
<td>97</td>
</tr>
<tr>
<td>Simple URL Rewriting Session Scheme</td>
<td>98</td>
</tr>
<tr>
<td>Wireless Device ID Session Scheme</td>
<td>101</td>
</tr>
<tr>
<td>Uses for Each Session Scheme</td>
<td>102</td>
</tr>
<tr>
<td>Multiple Session Schemes for Virtual Hosts</td>
<td>103</td>
</tr>
<tr>
<td>Deleting Attribute Cookies for Cookieless Federation</td>
<td>104</td>
</tr>
<tr>
<td>User Agent Settings in the Server.conf</td>
<td>104</td>
</tr>
<tr>
<td>Nokia User Agent Settings</td>
<td>105</td>
</tr>
<tr>
<td>Virtual Host Settings in the server.conf File</td>
<td>106</td>
</tr>
<tr>
<td>Setting Virtual Host Cookie Path and Domain to the Correct URI</td>
<td>107</td>
</tr>
<tr>
<td>Preserve the HOST Header File</td>
<td>108</td>
</tr>
<tr>
<td>Handling Large Files Using Data Blocks</td>
<td>108</td>
</tr>
<tr>
<td>Session Scheme Mapping for the Default Virtual Host</td>
<td>110</td>
</tr>
<tr>
<td>Web Agent Settings for the Default Virtual Host</td>
<td>111</td>
</tr>
<tr>
<td>Handling Redirects by Destination Servers</td>
<td>113</td>
</tr>
</tbody>
</table>
## Chapter 9: Configuring Proxy Rules

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Host Names Configuration</td>
<td>114</td>
</tr>
<tr>
<td>Override Default Values for Virtual Hosts</td>
<td>115</td>
</tr>
<tr>
<td>Proxy Rules Overview</td>
<td>117</td>
</tr>
<tr>
<td>Planning Routes for Incoming Requests</td>
<td>117</td>
</tr>
<tr>
<td>Proxy Rules Terminology</td>
<td>119</td>
</tr>
<tr>
<td>Establish a Proxy Rules Configuration File</td>
<td>120</td>
</tr>
<tr>
<td>Proxy Rules DTD</td>
<td>120</td>
</tr>
<tr>
<td>nete:proxyrules</td>
<td>121</td>
</tr>
<tr>
<td>nete:case</td>
<td>122</td>
</tr>
<tr>
<td>nete:cond</td>
<td>125</td>
</tr>
<tr>
<td>nete:default</td>
<td>127</td>
</tr>
<tr>
<td>nete:forward</td>
<td>128</td>
</tr>
<tr>
<td>nete:redirect</td>
<td>129</td>
</tr>
<tr>
<td>nete:local</td>
<td>129</td>
</tr>
<tr>
<td>nete:xprcond</td>
<td>130</td>
</tr>
<tr>
<td>How nete:xprcond Elements Works</td>
<td>132</td>
</tr>
<tr>
<td>Regular Expression Syntax</td>
<td>132</td>
</tr>
<tr>
<td>Regular Expression Examples in nete:rule and nete:result</td>
<td>135</td>
</tr>
<tr>
<td>Header Values in Forwards, Redirects, and Results Filters</td>
<td>136</td>
</tr>
<tr>
<td>Dynamic Header Value in a nete:forward</td>
<td>137</td>
</tr>
<tr>
<td>Dynamic Header Value in a nete:redirect</td>
<td>137</td>
</tr>
<tr>
<td>Dynamic Header Value in a nete:result</td>
<td>137</td>
</tr>
<tr>
<td>Response Handling</td>
<td>138</td>
</tr>
<tr>
<td>Modify Proxy Rules</td>
<td>138</td>
</tr>
<tr>
<td>Sample Proxy Rules Configuration Files</td>
<td>138</td>
</tr>
<tr>
<td>Proxy Rules Example—Routing Requests by Virtual Host</td>
<td>139</td>
</tr>
<tr>
<td>Proxy Rules Example—Routing Requests by Header Value</td>
<td>140</td>
</tr>
<tr>
<td>Proxy Rules Example—Routing Requests by Device Type</td>
<td>141</td>
</tr>
<tr>
<td>Proxy Rules Example—Routing Requests with URIs</td>
<td>141</td>
</tr>
<tr>
<td>Proxy Rules Example—Routing Requests by File Extension</td>
<td>142</td>
</tr>
<tr>
<td>Proxy Rules Example—Routing Requests with Nested Conditions</td>
<td>142</td>
</tr>
<tr>
<td>Proxy Rules Example—Using Regular Expression in Proxy Rules</td>
<td>143</td>
</tr>
</tbody>
</table>

## Chapter 10: Deploying the SPS

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPS Deployment in an Enterprise</td>
<td>145</td>
</tr>
<tr>
<td>Sticky-Bit Load Balancing</td>
<td>146</td>
</tr>
<tr>
<td>Proxying to Trusted Sites vs. Non-Trusted Sites</td>
<td>147</td>
</tr>
<tr>
<td>Configuring Virtual Hosts for the SPS</td>
<td>147</td>
</tr>
<tr>
<td>Implementing Session Scheme Mappings for Multiple Virtual Hosts</td>
<td>148</td>
</tr>
</tbody>
</table>
**Chapter 11: Integrating the SPS with SiteMinder**

- How the SPS Interacts with SiteMinder .......................................................... 153
- Authentication Scheme Considerations .......................................................... 154
- Proxy-Specific WebAgent.conf Settings ......................................................... 155
- Avoiding Policy Conflicts with Destination Server Web Agents ..................... 156
- Configuring SiteMinder Rules that Redirect Users ......................................... 158
- SPS and SharePoint Resources ...................................................................... 159
- SPS and ERP Resources ................................................................................. 159
- Password Services for SPS ............................................................................. 160
  - Configure a Password Policy for SPS ......................................................... 161
  - Verify Password Services for SPS .............................................................. 162
- Configuring Managed Self Registration for the SPS ..................................... 162
  - Install a Web Agent for MSR ................................................................... 163
  - Configure the Policy Server User Interface for MSR ................................. 164
  - Proxy Rules for an MSR Request ............................................................... 164
- Firewall Considerations .................................................................................. 165
- Keep Alive and Connection Pooling ............................................................... 165
- HTTP Header Configuration for Sun Java Web Servers ............................... 165
- HTTP Header for SiteMinder Processing with SPS ..................................... 166
- Handling Encoded URLs .............................................................................. 166

**Chapter 12: SSL and the Secure Proxy Server**

- Keys and Server Certificates Management ................................................. 167
  - Generate a Private RSA Key ................................................................... 168
  - RSA Key Decryption ................................................................................ 169
  - RSA Key Encryption ................................................................................ 169
  - Modify the Passphrase for an RSA Key ................................................... 169
  - Create Certificate Signing Request ......................................................... 170
  - Create a Self-Signed Certificate ............................................................... 170
  - Obtain Certificate Signed by a CA ............................................................ 171
  - Install a Signed Certificate ..................................................................... 171
- SSL Configuration for FIPS COMPAT and MIGRATE Modes ...................... 171
- SSL Configuration for FIPS ONLY Mode .................................................... 172
- Enable SSL for Virtual Hosts ...................................................................... 173

**Chapter 13: SPS APIs**

- Session Scheme API ....................................................................................... 175
  - Overview of Session Scheme API Processing ........................................... 175
  - Implement a Custom Session Scheme ....................................................... 179
  - Configure Rewritable Session Schemes ..................................................... 180
Use an IP Address Session Scheme ................................................................. 181
Session Storage API .................................................................................. 182
Filter API Overview .................................................................................. 182
How SPS Processes Custom Filters ............................................................ 183
Associate Custom Filters to Proxy Rules .................................................... 183
Filter API Class File .................................................................................. 184
ProxyFilter Interface ................................................................................ 184
BaseProxyFilter Abstract Implementation .................................................. 185
ProxyFilterConfig Interface ..................................................................... 187
ProxyResponse Interface .......................................................................... 188
ProxyFilterException Class ....................................................................... 189
ProxyRequest Interface ............................................................................ 190
Implement a Filter ..................................................................................... 192
Filter API Example ................................................................................... 192
Using a Filter to Rewrite Absolute Links in a Requested Page .................... 193

Chapter 14: Troubleshooting ................................................................... 195
Unable to Start Apache on UNIX systems .................................................. 195
DNS Caching in the SPS ............................................................................ 196
No Root Permissions ................................................................................ 197
Cannot Start the SPS Server ..................................................................... 197
Cannot Access the SPS with a Browser ..................................................... 198
Unknown Server Name ............................................................................ 198
Issues Configuring Virtual Hosts ............................................................... 199
Command not found Error Received ........................................................ 199
SPS Not Forwarding Requests .................................................................. 199
SPS and SharePoint .................................................................................. 200

Index ......................................................................................................... 201
Chapter 1: SiteMinder Secure Proxy Server Overview

This section contains the following topics:

- Introduction to the Secure Proxy Server (see page 11)
- Secure Proxy Server Product Features (see page 16)
- Secure Proxy Server Product Limitations (see page 17)
- SPS in an Enterprise (see page 18)

Introduction to the Secure Proxy Server

The SiteMinder Secure Proxy Server (SPS) is a stand-alone server that provides a proxy-based solution for access control. The SPS employs a powerful proxy engine that provides a network gateway for the enterprise and supports multiple session schemes that do not rely on traditional cookie-based technology. This guide discusses how to configure and deploy the SPS in an enterprise.

Proxy Server Architecture

A traditional proxy server is located between a firewall and an internal network and provides caching of resources and security for the users on the internal network. Traditional proxy servers act as a proxy on behalf of a group of users for all resources on the Internet.

The following illustration shows a proxy server configuration. The proxy server caches frequently accessed resources so that requests for those resources are handled faster in the Demilitarized Zone (DMZ).
**Traditional Reverse Proxy Server Architecture**

A reverse proxy server represents one or more destination servers. A typical use of reverse proxy architecture provides:

- Cached resources
- Security
- SSL acceleration
- Load balancing

Rather than requesting a resource directly from a destination server, the reverse proxy server caches much of the content from the destination servers, providing ready access for users. This type of proxy server deployment is considered a reverse proxy, because the proxy is transparent to the user and works on behalf of the destination servers in the enterprise. Multiple reverse proxy servers can be used for load balancing and can also provide some SSL acceleration for HTTPS requests. A reverse proxy server also provides an additional layer of security by insulating destination servers that reside behind the DMZ.

**Secure Proxy Server Architecture**

The SPS is not a traditional reverse proxy solution, because it does not provide resource caching. The SPS does serve, however, as a single gateway for access to enterprise resources, regardless of the method of network access.

A set of configurable proxy rules determines how the SPS handles a user request. Users can access resources through multiple session schemes based on mapping between user agent types and virtual hosts. Requests can be routed to different destination servers based on the type of device being used to access the network.
The following illustration shows a configuration of the SPS. Users access the SPS using various devices. The SPS determines the session scheme to create based on the access device, and then forwards or redirects requests to the appropriate destination servers. Users are not aware that there is a reverse proxy server in the enterprise.
A stand-alone SPS consists of an HTTP listener (Apache) and a Tomcat servlet container, as shown in the following illustration:

**Java Web Agent Components**

- **Java Web Agent**
- **JNI Wrapper**
- **Servlet/Generic filter context**
- **C++ Agent framework**

---

**SPS Components**

A stand-alone SPS consists of an HTTP listener (Apache) and a Tomcat servlet container, as shown in the following illustration:
The SPS architecture includes the following components:

**Apache**

SPS uses the open source Apache Web server to act as the HTTP listener for incoming requests. An additional component, mod_jk (1.2.18), acts as the Tomcat connector, which enables communication between the Apache Web server and Tomcat using the Apache JServ protocol (AJP).

**Tomcat**

The Tomcat server provides Tomcat servlet container for the SPS. The Tomcat initialization is customized so it does not allow deployment of any external applications or servlets. The standard Tomcat xml (server.xml) is not used for initialization. The components inside the Tomcat container of the SPS include the following:

**Configuration Resolver ProxyBootstrap**

The configuration resolver proxybootstrap is responsible for reading the SPS configuration from the server.conf file and initializes the SPS.

**Session Discovery**

The session discovery component analyzes the incoming requests for extracting the SPS session information. Depending on the user agent type and the virtual host being used, this component uses the appropriate session scheme for extracting the SPS session information.

If the request uses an existing SPS session, this component uses the SPS session identifier contained in the request to extract the corresponding SiteMinder session from the in-memory session store. The SPS passes the SiteMinder session to the Java Web Agent for session validation. If the request does not contain an existing SPS session, this component passes the request on to the Java Web Agent for user authentication.

**Java Web Agent**

The Java Web Agent, together with the SiteMinder Policy Server, authenticates and authorizes the user.

**Post Agent Session Writer**

The post Agent session writer performs additional processing for cookieless session schemes. After the Java Web Agent authenticates and authorizes the user and creates a SiteMinder session, this component creates an SPS session identifier. This identifier is attached to the SiteMinder session created by the Java Web Agent.

This session identifier is then maintained in the in-memory session store of the SPS. In addition to maintaining the session in the session store, this component transforms the URI. For example, the Post Agent Session Writer manipulates the URI for the simple_url session scheme.
Proxy Rules Servlet Filter

The proxy rules servlet filter loads the proxy rules from the proxyrules.xml file. Depending upon the incoming request and the proxy rule, the request is forwarded or redirected to the backend server. If the request is forwarded, an open source component Noodle is used.

Any changes made to the proxy rules do not require a restart for the changes to take effect. The proxyrules are reloaded when there is any change in the proxyrules.xml file.

Noodle Servlet

The Noodle servlet forwards requests to the backend servers. Noodle also supports the use of proxy pre-filters which enable the request to be modified before sending the same to the backend server. Similarly support for proxy post-filters is also available which enables modification of the response received from the backend server before sending it back to the user client.

HTTP Client

The HTTP client sends requests to the backend server and receives responses from the backend server.

Secure Proxy Server Product Features

The SPS offers the following features:

Access Control for HTTP and HTTPS Requests

SPS allows you to control the flow of HTTP and HTTPS requests to and from destination servers using an embedded SiteMinder web agent. In addition, the SPS is fully integrated with SiteMinder to manage e-business transactions.

Single Sign-on

The embedded web agent in the SPS enables single sign-on (SSO) across an enterprise, including SSO with SiteMinder Web agents that can be installed on destination servers within the enterprise.

Multiple Session Schemes

A session scheme is a method for maintaining the identity of a user after authentication. Core SiteMinder products use cookies to maintain a session. The SPS, however, can maintain sessions based on SSL ID, mini-cookies, device IDs for handheld devices, URL rewriting, IP addresses, and schemes created using the Session Scheme API.
Secure Proxy Server Product Limitations

The following conditions apply to the SPS:

- SPS is not a plug-in to another Web server. The SPS is a fully supported, stand-alone server.
- The SPS does not support local content. The ability to place content on the SPS is not exposed, and the SPS does not support proxy rules for providing access to local content.
- SPS does not support having the Web server on the same system as the SPS. If the two are set up on the same system, the Web server is accessible from the Internet. Security is not sure with this configuration.
- The SPS provides its own session storage. However, the session store has no public APIs for use as a general session server.
In some enterprises that use the SPS, destination servers can also have SiteMinder web agents or application server agents installed. When policies for the SPS agent are inconsistent with policies for the agent installed on the destination server, the SPS can pass responses back to the invoking client. Because the SPS does not provide warnings about inconsistencies in processing such policies, use care when setting up SiteMinder policies in such environments.

- The SPS makes a new request to the destination server for every request that it receives. All caching directives are ignored.
- In the simple-url session scheme, SPS does not rewrite URLs embedded in or resulting from JavaScript.
- The simple_url session scheme does not preserve the POST data when posting to a protected resource.

**SPS in an Enterprise**

Enterprises that provide access to network resources for employees, customers, and partners face a number of challenges, including:

- Directing requests to appropriate services
- Verifying user identities and establishing entitlements
- Maintaining sessions for authorized users
- Providing centralized access control configuration
- Supporting multiple device types
- Employing flexible and secure architectures

SiteMinder provides solutions to many of these challenges, including authentication and authorization of users, and a complex engine for evaluating user entitlements. The SPS further expands the benefits of core Policy Server and Web Agent functionality by providing a reverse proxy solution.

This reverse proxy solution adds the following capabilities:

- Inter-operability with existing SiteMinder Web Agents
- Cookieless single sign-on and sessions storage
You can deploy the SPS in an enterprise to serve the following functions:

- Act as a centralized access control filter
- Support cookieless session schemes
- Support extranet access control

**SPS as a Centralized Access Control Filter**

To limit access to destination servers and provide a central entry point to the network, the SPS can be placed in front of all destination servers in the enterprise. HTTP or HTTPS requests that come into the enterprise can be filtered through the SPS, and forwarded to the appropriate destination server for fulfillment.

The following illustration shows how the SPS handles all HTTP and HTTPS requests.
Destination servers that contain content do not require SiteMinder Web Agents. The only network element that resides behind the first firewall is the SPS. All users must be authenticated and authorized by SiteMinder residing behind the second firewall. The destination servers provide content after SiteMinder and the SPS verify user entitlements.

This deployment provides the following benefits:

- Centralizes configuration through proxy rules
  The SPS uses proxy rules defined in XML configuration files to establish how the SPS handles requests. Proxy rules can be based on:
  - Host name
  - URI substring
  - HTTP header
  - SiteMinder header
  - Regular Expressions based on URI
In addition, the conditions for proxy rules can be nested to create rules that incorporate multiple conditions.

- Directs requests to appropriate services
  All HTTP and HTTPS traffic passes through the SPS. Based on the proxy rules established for the SPS, requests are forwarded to the appropriate destination servers for fulfillment.

- Verifies user identities and establishes entitlements
  The SPS uses the built-in web agent to communicate with SiteMinder and perform authentication and authorization of requests.

### SPS Support for Cookieless Sessions

Most solutions use cookie technology. However, when accessing resources over HTTP or HTTPS, some enterprises want alternatives for establishing and maintaining a user session and provide single sign-on with a cookieless solution.

The SPS provides an in-memory session store and allows the use of any of the following cookieless session schemes:

- Mini-cookies (uses small cookies in place of standard SiteMinder cookies)
- SSL ID
- Device ID
- Simple URL Rewriting
- IP Address
The following illustration shows a deployment in which the SPS provides a combination of standard sessions using cookies and sessions without cookies:

- **Supports multiple device types**

  Through a set of proxy rules, the SPS forwards, or redirects, requests based on the type of device issuing the requests. For example, all initial requests can be directed at the SPS, which forwards requests to destination servers based on device types. Browser requests can be redirected to destination servers, and the SPS handles wireless requests.

- **Maintains sessions for authorized users**

  Both standard SiteMinder cookies and cookieless session schemes are employed for maintaining user sessions. Session schemes are assigned based on user agent type for each virtual host. For example, all users accessing the network through web browsers are assigned to a standard cookie session scheme. Users accessing resource through a wireless telephone are assigned to a device ID session scheme.
Provides cookieless single sign-on and session storage

Through an in-memory session store and the support of multiple session schemes, the SPS provides alternatives to cookie-based sessions. The SPS maintains session information in the session store and returns a token. This token is exchanged with all transactions, allowing the SPS to match the token to the session information captured in the session store.

Offers multiple options for maintaining sessions

**Cookieless Session Scheme in a Federation Environment**

The SPS, with its built-in handling of cookieless session schemes, enables it to be deployed in environments where the user agent, such as a wireless device, does not support traditional SiteMinder cookies.

If you deploy the SPS in a SiteMinder federation security services environment, the following process is enforced when a user request is received:

1. The SPS receives a request for a federated resource. The request is directed to the Federation Web Services (FWS) application at the site producing assertions.

2. The SPS verifies if cookieless federation is enabled for the virtual host requesting the redirect.

3. If a cookieless scheme is being used, the SPS removes the session key (SMSESSION cookie) for the current session.

4. The SPS sends the user to the link provided by the FWS redirect.

If the SPS is using a rewritable session scheme such as simple_url session scheme, the SPS rewrites the redirect response to include the session key information in the redirected URL.

**SPS Support for Extranet Access Control**

Another deployment of the SPS provides access control for external users, but allows direct access to destination servers for internal users. If a destination server provides access to secure applications for individuals within the enterprise, a standard SiteMinder Web Agent can be installed on the destination server to provide access control. Users who are properly authenticated through the SPS can use single sign-on.
The following illustration shows an example of an extranet network deployment.

This deployment provides the following benefits:

- Directs requests from extranet sources
  All extranet traffic passes through the SPS and is forwarded to the appropriate destination server once users are authenticated and authorized for requested resources.

- Employs flexible architectures
  All information is located behind multiple firewalls to protect from extranet attacks. Information that is appropriate for intranet users does not incur the overhead of agent to SiteMinder communication. SiteMinder can still protect sensitive resources, however.

- Provides interoperability between Web Agents
  The SPS and intranet Web Agents use the same Policy Server and provide single sign-on for authorized extranet users on all destination servers.
Chapter 2: Installing and Uninstalling the Secure Proxy Server

This section contains the following topics:

Before You Install (see page 25)
Install the SPS on Windows (see page 27)
Install the SPS on UNIX (see page 28)
Cancel the SPS Installation (see page 28)
Run the Configuration Wizard (see page 29)
Start and Stop the SPS (see page 30)
Start the SPS in Single or Multi Process Mode (see page 31)
Confirm that the Secure Proxy Server is Functioning (see page 32)
Reinstall the SPS (see page 32)
Silent Installation and Configuration (see page 33)
Uninstall the SPS from UNIX (see page 33)
Uninstall the SPS from Windows (see page 34)
Additional Configuration on the SPS (see page 34)
Modify the Default Location of the SiteMinder Forms (see page 35)

Before You Install

To use the SPS, the system where you plan to install it must have at least 256 MB of RAM. Other prerequisites differ based on the server system.

For detailed information, see the SPS Support Matrix at http://ca.com/support.

Note: Installation prerequisites pertain to the system on which you run the SPS, not the destination servers to which the SPS routes incoming requests.

The SPS installation consists of two tasks:
1. Install the software.
2. Run the configuration tool.

Note: Throughout the installation instructions, there are references to sps_home in directory paths. This variable represents the installation directory of the SPS.

During installation, you are prompted for specific information. We recommend having this information ready before you run the SPS installation.
Requested Host Registration Information

The SPS configuration wizard displays a series of prompts for registering a trusted host. A trusted host is a client computer where one or more SiteMinder Web Agents can be installed. The term trusted host refers to the physical system. To establish a connection between the trusted host and the Policy Server, register the host with the Policy Server. After registration is complete, the registration tool creates the SmHost.conf file. When this file is created successfully, the client computer becomes a trusted host.

The following table lists the required host registration information:

<table>
<thead>
<tr>
<th>Information Requested</th>
<th>Description/Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>SiteMinder administrator name</td>
<td>Name of the administrator that matches the name already defined at the Policy Server. This administrator must have the privileges to create a trusted host.</td>
</tr>
<tr>
<td>SiteMinder administrator password</td>
<td>Password of the SiteMinder administrator who has privileges to register a trusted host. Must match the password used at the Policy Server.</td>
</tr>
<tr>
<td>Trusted host name</td>
<td>Name of the trusted host assigned during the installation.</td>
</tr>
<tr>
<td>Host Configuration Object</td>
<td>Name of a host configuration object already defined in the Policy Server administrative UI.</td>
</tr>
<tr>
<td>Agent Configuration Object</td>
<td>Name of an existing Agent Configuration Object defined in the Policy Server administrative UI</td>
</tr>
<tr>
<td>IP address of the Policy Server where the host is registered</td>
<td><strong>Note</strong>: Include a port number when SiteMinder is behind a firewall. For example, 111.12.12.2:12</td>
</tr>
<tr>
<td>Host Configuration File name and location</td>
<td>Identifies the SmHost.conf file, which Web Agents and custom Agents use to act on behalf of the trusted host. Using this file, the host can find a Policy Server and establish a connection. The wizard lists the default location.</td>
</tr>
<tr>
<td>Name and location of the Web Agent configuration file</td>
<td>WebAgent.conf Default location provided by wizard</td>
</tr>
</tbody>
</table>
Requested Apache Web Server Information

The SPS configuration wizard displays a series of prompts for the embedded Apache web server. The following table lists the information required by the wizard:

<table>
<thead>
<tr>
<th>Information Requested</th>
<th>Description/Default</th>
</tr>
</thead>
</table>
| Email address of the Apache web server administrator | The email address for the administrator  
Default: admin@company.com |
| Fully qualified host name of the server         | A fully qualified name in the following format:  
computer_name.company.com                        |
| Port number for HTTP requests                   | The port listening for HTTP requests  
Default: 80                                      |
| Port number for SSL requests                    | The port listening for SSL requests  
Default: 443                                      |

Install the SPS on Windows

The SPS can run on Windows 2008. The default installation location for the SPS on Windows systems is: C:\Program Files\CA\secure-proxy.

Note: The installation fails if you install the SPS on a FAT file-system partition.

To install the SPS on Windows

1. Copy the installation program from the download location on the CA Support site.
2. Right-click the executable and select Run as administrator.
   The installation program starts.
4. Follow the instructions from the installation wizard.
5. Restart your system after the installation finishes.

To view the record of the installation, go to the directory sps_home\install_config_info and look at this log file:

CA_SiteMinder_Secure_Proxy_InstallLog.log
Install the SPS on UNIX

The SPS supports installations on Linux and Solaris. For a list of required and recommended Solaris patches and required RPM packages for Linux, see the SPS Release Notes located in the installation directory.

The default installation location is user_home/CA/secure-proxy. The folder where you install the SPS must have sufficient permissions (755). Do not install the SPS under the /root folder, because its default permissions (750) are insufficient.

On Solaris, SPS runs as the "nobody" user. If you prefer not to run SPS as this user, create an alternate user and assign the necessary permissions.

To install the SPS on UNIX

1. Copy one of the following programs from the download location on the CA Support site to a temporary directory:
   - Solaris: ca-proxy-12.0-sp3-sol.bin
   - Linux: ca-proxy-12.0-sp3-rhel30.bin

2. Enter one of the following commands:
   - sh ./ca-proxy-12.0-sp3-sol.bin
   - sh ./ca-proxy-12.0-sp3-rhel30.bin

3. Follow the screen prompts provided by the installation wizard.

   To determine whether the installation was successful, go to the directory sps_home/install_config_info and look at the log file CA_SiteMinder_Secure_Proxy_Server_InstallLog.log

Cancel the SPS Installation

You can cancel an SPS installation at any time during the process.

To cancel an installation

1. Click Cancel in any of the installation dialogs
   If you are running a console mode installation on UNIX, you can type quit at the command line.

2. If you cancel while installation files are written to your system, remove any files that have been already been written to your system.

   The installation stops.
Run the Configuration Wizard

After you install the SPS, run the configuration wizard. The configuration wizard takes you through the process of trusted host registration for the embedded SiteMinder Web Agent and performs some administrative tasks for the embedded Apache web server.

**Important!** Before you run the wizard, verify that you have set up the required objects at the Policy Server where you want to register the host. If these objects are not configured, trusted host registration fails.

To run the configuration wizard

1. Open a console window and navigate to the directory `sps_home/secure-proxy`.
2. Enter one of the following commands:
   - Windows: `ca-sps-config.exe`
   - UNIX: `ca-sps-config.sh`
   
   The wizard starts.
3. Select the option to perform host registration immediately.
   This screen also includes an optional check box for enabling shared secret rollover.
4. As part of the trusted host registration process, respond to the prompts as follows:
   a. Specify the name and password of the SiteMinder administrator. The information you enter must already be defined at the Policy Server where the trusted host is registered.
   b. Specify the name of the Trusted Host and the Host Configuration Object.
      The name you enter for the trusted host must be unique. The name for the Host Configuration Object must already be defined at the Policy Server where the trusted host is registered.
   c. Enter the IP address of the Policy Server where you want to register the trusted host.
   d. Specify the name and location of the host configuration file, `SmHost.conf`. The wizard lists the default location.
   e. Specify the name of the Agent Configuration Object. The Agent Configuration Object that you enter must already be defined at the Policy Server where the trusted host is registered.
5. Enter the following information for the Apache web server:
   ■ Server name
   ■ Web server administrator email address, in the form admin@computer_name.com
   ■ HTTP port number. The default is 80.
   ■ HTTPS (SSL) port number. The default is 443.

   **Note:** Users installing on systems running Solaris or Linux see an additional screen that prompts for the name of the user under which Tomcat and Apache runs. This user cannot be root. Create the user account manually; the installation program does not create it for you. The Tomcat user must have all privileges (rwa) for the log directories.

6. Select Yes if you want the SPS to act as a Federation Gateway.

7. Review the Configuration Summary

8. Click Install.
   The files are installed.

9. Click Done to exit the wizard.

   **Note:** If you run the Configuration Wizard again for any reason, SSL must be reinitialized.

---

**Start and Stop the SPS**

Use the procedures listed following to start and stop SPS.

**To start or stop the SPS on a system running Windows**

1. From the Windows Start menu navigate to Settings, Control Panel, Administrative Tools, Services.
2. Scroll down the list of services and select SiteMinder Secure Proxy.
3. From the Action menu, select All Tasks and select the command you want.
4. Repeat Step 3 for SiteMinder proxy engine.

   **Note:** You must be logged on as root user to run the commands listed following.

**To start the SPS on a system running UNIX**

1. Navigate to the following directory:
   ```
   sps_home/proxy-engine
   ```
2. Run the following from a command line:
   ```
   .sps-ctl start
   ```
To stop the SPS on a system running UNIX

1. Navigate to the following directory:
   sps_home/proxy-engine
2. Run the following from a command line:
   ./sps-ctl stop

To start the SPS in SSL mode on a system running UNIX

1. Navigate to the following directory:
   sps_home/proxy-engine
2. Stop the SPS.
3. Enter the following command from at a command-line prompt:
   ./sps-ctl startssl

Start the SPS in Single or Multi Process Mode

The SPS supports single-process and multiple-process modes, which enable the embedded Web Agent to create the Low Level Agent Worker Process (LLAWP) at runtime.

The SPS can be configured to start in a single-process or multiple-process mode. Single-process mode is the default.

The modes operate as follows:

**Single-Process Mode**

This mode causes the Web Agent to use local resources rather than shared operating system resources offered by the LLAWP to operate. No separate LLAWP processes are started in single-process mode. When multiple virtual hosts run, single-process mode results in an increase in the memory footprint of the SPS Java process.

*Note:* Single-process mode is supported only for host servers that run as a single server process.

**Multiple-Process Mode**

This mode causes the LLAWP framework to spawn a process for every virtual host. Because multiple-process mode uses shared memory, the SPS uses shared operating system resources for logging, caching, and monitoring of multiple web server processes.
To set the mode of operation
1. Open the server.conf file in a text editor.
2. Set the singleprocessmode parameter as follows:
   - To use single process mode, keep singleprocessmode set to yes.
   - To use multiple-process mode, change singleprocessmode to no.
3. If you modify the server.conf file, restart SPS.

The SPS mode of operation is set.

Confirm that the Secure Proxy Server is Functioning

After you install the SPS, but before changing the proxy rules, you can verify that the server is functioning. You can request the index.html by using the server and port number you specified during installation. For example, if you installed the SPS on server1.company.com and selected port 88 for HTTP communication, you can request the following from a browser:

http://server1.company.com:88/

If the SPS is working properly, control shifts to the main CA website (www.ca.com). This site is configured as the default in the simple proxy rules configuration file.

Reinstall the SPS

You can reinstall or reconfigure the SPS. You do not have to remove the existing installation.

To reinstall and reconfigure the SPS
1. Rerun the installation program, ca-proxy-12.0-sp3-operating_system.
   
   operating_system  
   
   Defines the operating system, either win32, sol, or rhel30.
2. Run the configuration wizard ca-sps-config again.

Reinstallation of the SPS is complete.

More information:

Install the SPS on Windows (see page 27)
Install the SPS on UNIX (see page 28)
Run the Configuration Wizard (see page 29)
Silent Installation and Configuration

After you have installed and configured the SPS for the first-time, you can reinstall it unattended at a later time. Or, you can install another instance of the SPS unattended using saved configuration data.

After installation, the SPS creates a sample properties file in the sps-home/install_config_info folder. After configuration, the same properties file is updated with additional properties for configuration. This properties file is used for subsequent silent installation and configuration with customized values.

To install or configure the SPS unattended

1. Open a command window.
2. Navigate to the folder where you installed the properties file. The default is sps-home/install_config_info.
3. Enter one for the following commands at the prompt:

   For silent installation
   ```
   ca-proxy-12.0-sp3-operating_system -i silent -f ca-sps-installer.properties
   ```
   
   ```
   operating_system
   ```
   Defines the operating system, either win32, sol, or rhel30.

   For silent configuration
   ```
   ca-sps-config-i silent -f ca-sps-installer.properties
   ```

   The installation or configuration proceeds without further user interaction.

Uninstall the SPS from UNIX

Use the following procedure to uninstall SPS from a UNIX system.

To uninstall SPS from a UNIX system

1. Open a console window.
2. Navigate to the root installation directory.
3. Run the following program at the command prompt:

   ```
   ./ca-sps-uninstall.sh
   ```

   **Note:** If you have modified any SPS files, such as server.conf, the uninstall program does not remove these files or their parent folders automatically. Remove any files and folders for files you have changed.
Uninstall the SPS from Windows

Use the following procedure to remove the SPS from your system.

**To remove the SPS**

1. Select Settings, Control Panel from the Windows Start menu.
2. Select Add/Remove Programs.
3. Select CA SiteMinder Secure Proxy Server 12.0 SP3.
4. Click Change/Remove.
5. Read the confirmation information and click Uninstall.
6. Click Finish.

The SPS is removed from your system.

**Note:** If you have modified any of the SPS files such as server.conf, the uninstall program does not remove these files or their parent folders automatically.

Additional Configuration on the SPS

After installing the SPS and running the configuration wizard, you can modify the SPS configuration to suit your environment. The following configuration files contain settings that affect the SPS:

**httpd.conf**
Contains the settings for the Apache web server.

**server.conf**
Contains the settings that determine the SPS behavior, including logging, virtual hosts, and session scheme mapping.

**proxyrules.xml**
Contains the rules that determine how the SPS handles incoming requests.

**More information:**

- [Configuring the Apache Web Server](#) (see page 71)
- [SPS server.conf File Overview](#) (see page 73)
- [Configuring Proxy Rules](#) (see page 117)
Modify the Default Location of the SiteMinder Forms

Beginning with SPS v6.0, the default location of the SiteMinder forms is no longer /siteminderagent/forms. To continue to use this location to serve forms, modify the forms location for the SPS.

To modify the forms location
1. Create the siteminderagent directory in the following location:
   
   $sps\_home/proxy$-engine/examples/siteminderagent

2. Copy the forms folder from the following directory
   
   $sps\_home/proxy$-engine/examples
   
   to the following directory:

   $sps\_home/proxy$-engine/examples/siteminderagent

   The forms are copied to $sps\_home/proxy$-engine/examples/siteminderagent/forms.
Chapter 3: Upgrading the Secure Proxy Server

This section contains the following topics:

SPS Upgrade Overview (see page 37)
Upgrading from SPS v6.0 (see page 37)
Additional Tasks for Upgrades (see page 38)

SPS Upgrade Overview

You can upgrade from a previous version of the SPS by running the installation program on a system where the SPS is already installed. When you begin the installation, the program asks whether you want to upgrade. After confirmation, the installation proceeds.

Important! If you customized your existing SPS deployment, make the same modifications to the upgraded SPS configuration so that it can operate in your environment.

More information:
Installing and Uninstalling the Secure Proxy Server (see page 25)

Upgrading from SPS v6.0

Run the Configuration Wizard after installing or upgrading to SPS r12.0 SP3, because the Apache server uses different configuration files. Earlier versions of the SPS implemented a different version of Apache.

Restart the SPS services after any change to the Apache configuration file (these changes are not reflected automatically in a new build).

If you run the Configuration Wizard for a second time for an upgrade, the wizard automatically disables the SSL settings, so you must follow the manual steps to enable SSL. In addition, be sure that the SSLCertificateFile listed in server.conf is located in the specified path.

More information:
SSL Configuration for FIPS COMPAT and MIGRATE Modes (see page 171)
Additional Tasks for Upgrades

At the end of the installation process, you can perform some additional steps to support the upgrade. Depending on the amount of customization in your SPS deployment, you can perform one or more of the following tasks:

- Verify that the SSL configuration paths inside the ssl.conf file and the server.conf file are correct for your environment. The automated portion of the upgrade assumes that all certificates are in the default location.

- Verify that all certificates, Certificate Authorities, and keys have been correctly copied to their folders in the `sps_home\secure-proxy\SSL`.

- Modify the path to the proxy rules DTD file in the proxyrules.xml file. The default path of the DTD file is `sps_home\proxy-engine\conf\dtd\proxyrules.dtd`.

- If you are upgrading from SPS r6.0 to SPS r12.0 SP 3, add the following parameter to the `<VirtualHostDefaults>` section in the server.conf file before running the upgrade:
  
enablerewritecookiepath="no"

  This addition is done automatically when upgrading from SPS r6.0 SP 1 or later.

Duplicate Custom Settings

If you have created any custom settings in your current installation, duplicate them in the new one. For example, if you have configured additional virtual hosts, add these additional configuration directives to the httpd.conf file.

To duplicate custom settings

1. Modify the httpd.conf file.

2. Modify the ssl.conf file.

   **Note:** Since release r6.0 SP3, the ssl.conf file has been renamed and moved to `sps_home\httpd\conf\extra\httpd-ssl.conf`.

3. Modify the server.conf file.

   **Note:** During the upgrade, the existing server.conf file is backed up to a file named `server.conf-date.bak` in the `sps_home\secure-proxy\proxy-engine\conf` directory.

4. Copy the existing custom session schemes and filter class files to the new installation.

5. Deploy any custom Java class or .jar files related to the SPS filter or session scheme APIs.
Customize JVM Parameters

You can customize Java Virtual Machine (JVM) parameters in the following files:

- On Windows, modify the SmSpsProxyEngine.properties file located in the directory `sps_home\proxy-engine\conf`.
- On UNIX, modify the proxyserver.sh file located in the directory `sps_home/proxy-engine`.
Chapter 4: FIPS-140 Support

This section contains the following topics:

- **FIPS Support Overview** (see page 41)
- **Configuration Process for FIPS ONLY Mode** (see page 42)
- **Migration to FIPS MIGRATE Mode** (see page 42)
- **Migration to FIPS ONLY Mode** (see page 43)

**FIPS Support Overview**

The Secure Proxy Server supports the requirements for cryptographic modules specified in the FIPS 140-2 standard. When you install SPS, a dialog appears that prompts you to select the level of FIPS support your operating configuration requires.

During a new installation you can select one of these three FIPS modes:

- **COMPAT** — Specifies that the installation is not FIPS-compliant. Select this mode when interacting with clients running earlier versions of the SPS.
- **MIGRATE** — Specifies that the SPS operates both with FIPS-compliant algorithms and algorithms used in earlier version of the SPS simultaneously while the data is migrated.
- **ONLY** — Specifies that only FIPS-compliant algorithms are used and accepted by the SPS. When you install in this mode, additional manual configuration is required.

The FIPS mode you select during installation usually is the same as the FIPS mode configured on the Policy Server. When the Policy Server is in Migrate mode, it can operate with the SPS in any mode.

If you are upgrading an existing SPS installation to SPS r12 SP 3, the SPS continues to work as before, that is, in COMPAT mode. You can change the mode manually using the smreghost command, as described in subsequent sections. Be sure to restart the system after a mode change so that the Web Agent, the SPS server, and the Apache server pick up the changes.

**More information:**

- **Migration to FIPS MIGRATE Mode** (see page 42)
- **Migration to FIPS ONLY Mode** (see page 43)
- **SSL Configuration for FIPS ONLY Mode** (see page 172)
- **Configuration Process for FIPS ONLY Mode** (see page 42)
Configuration Process for FIPS ONLY Mode

After you install the SPS in FIPS ONLY mode, the following additional configuration steps are required:

- Verify that the SPS is running in full SSL mode.
- Verify that the server key used to configure the SPS in SSL mode was generated using a FIPS-compliant cryptographic algorithm.
- Follow the procedure for configuring SSL in FIPS ONLY mode.

More information:

SSL Configuration for FIPS ONLY Mode (see page 172)

Migration to FIPS MIGRATE Mode

If you are upgrading to SPS r12 SP 3 from an earlier version and want to use FIPS-compliant algorithms, you can change the Web Agent inside the SPS from COMPAT mode to MIGRATE mode.

To set the SPS to FIPS MIGRATE mode

1. Stop the SPS services.
2. Open a command-line window.
3. Enter the following command:
   
   ```
   smreghost -i policy_server_ip_address -u administrator_user_name -p administrator_password -hn hostname_for_registration -hc host_config_object -f path_to_host_config_file -o -cf MIGRATE
   ```
   
   Example:
   
   ```
   smreghost -i localhost -u siteminder -p firewall -hn helloworld -hc host -f "C:\Program Files\CA\secure-proxy\proxy-engine\conf\defaultagent\SmHost.conf" -o -cf MIGRATE
   ```
4. Restart the SPS machine. (Windows only)
5. Restart the SPS services.

The Web Agent inside the SPS is changed from FIPS COMPAT to FIPS MIGRATE mode.
Migration to FIPS ONLY Mode

On an upgrade, you can change the FIPS mode on the SPS from COMPAT to ONLY as long as the SiteMinder Policy Server is also in FIPS ONLY mode or FIPS COMPAT mode.

To change the SPS to FIPS Only mode

1. Stop SPS services.

2. Set the OPENSSL_FIPS environment variable with a value of 1.
   This setting enables FIPS mode for the openssl command-line utility.

3. Set the CA_SM_PS_FIPS140 environment variable with a value of ONLY.
   This setting enables SPS and Apache code determine the FIPS mode.
   For UNIX
   In the proxyserver.sh file at sps-home/proxy-engine/proxyserver.sh set the CA_SM_PS_FIPS140 environment variable to a value of ONLY.

4. Open a command-line window.

5. Run the following command:

   smreghost -i policy_server_ip_address -u administrator_user_name -p administrator_password -hn hostname_for_registration -hc host_config_object -f path_to_host_config_file -o -cf ONLY
   Example:
   smreghost -i localhost -u siteminder –p firewall -hn helloworld -hc host -f "C:\Program Files\CA\secure-proxy\proxy-engine\conf\defaultagent\SmHost.conf" -o -cf ONLY

6. Determine whether the SPS is running in full SSL mode. If SSL is already enabled on Apache inside SPS, SSL must be disabled and reconfigured for FIPS ONLY mode.

7. Change the value of the SSLPassPhraseDialog variable in httpd-ssl.conf (present in sps_home\httpd\conf\extra folder) from builtin to custom.

8. Uncomment the following line in httpd-ssl.conf:

   SSLCustomPropertiesFile "<sps_home>/Tomcat/properties/spsssl.properties"

9. Restart the SPS machine. (Windows only)

10. Start SPS services.
Migration to FIPS ONLY Mode

More information:

Configuration Process for FIPS ONLY Mode (see page 42)
Chapter 5: Using the SPS with Federation Security Services

This section contains the following topics:

- **Federation Security Services Introduction** (see page 45)
- **SPS Use Cases in a SiteMinder Federated Environment** (see page 45)
- **SPS Roles in a SiteMinder Federated Environment** (see page 50)
- **Solutions for SPS Use Cases** (see page 50)
- **Cookieless Federation** (see page 59)
- **SPS as a Web Agent Replacement** (see page 61)
- **SPS as a Federation Gateway** (see page 63)

## Federation Security Services Introduction

SiteMinder Federation Security Services (FSS) allow the exchange of security information between business partners. The services provide seamless authentication and fine-grained authorization across enterprises.

Federation Security Services are implemented with the SPS in the following ways:

- As a replacement for a SiteMinder Web Agent.
- As a replacement for the SiteMinder Web Agent and the Web Agent Option Pack.

Federation services enable an organization and its partners to:

- Exchange user information in a secure manner
- Map user identities at one organization to user identities at other organizations
- Provide single sign-on across different organizations
- Control access to resources based on user information received from a partner
- Interoperate across heterogeneous environments, such as Windows, UNIX and various Web servers, such as IIS, Sun Java System and Apache

## SPS Use Cases in a SiteMinder Federated Environment

There are probably as many use cases for federated networks as there are business arrangements between partners. The use cases that follow demonstrate different ways of handling user identities to provide single sign-on between partners.

For more use cases, see the CA SiteMinder Federation Security Services Guide.
Use Case 1: Single Sign-on Based on Account Linking

In Use Case 1, smcompany.com contracts with a partner company, ahealthco.com to manage employee health benefits.

An employee of smcompany.com authenticates at an employee portal at his company’s site, www.smcompany.com and clicks a link to view her health benefits at ahealthco.com. The employee is taken to ahealthco.com’s web site and is presented with her health benefit information without having to sign on to ahealthco.com’s Web site.

The following illustration shows this use case.

The company, ahealthco.com, maintains all health-related information for employees at smcompany.com. To do this, ahealthco.com maintains user identities for every employee of smcompany.com. When an employee of smcompany.com accesses ahealthco.com, an identifier for the employee is passed from smcompany.com to ahealthco.com in a secure manner. This identifier allows ahealthco.com to determine who the user is and the level of access to allow for that user.

More information:

Solution 1: SSO Based on Account Linking (see page 50)
Use Case 2: Single Sign-on Based on User Attribute Profiles

In Use Case 2, smcompany.com buys parts from a partner named partsco.com.

An engineer authenticates at his employee portal, smcompany.com and clicks a link to access information at partsco.com. Because the user is an engineer at smcompany.com, he is taken directly to the Specifications and Parts List portion of partsco.com's web site without having to sign in.

When a buyer for smcompany.com authenticates at smcompany.com and clicks a link to access information at partsco.com, she is taken directly to the ordering area of partsco.com's web site without having to sign on.

Additional attributes, such as user name are passed from smcompany.com to partsco.com to personalize the interface for the individual user.

Partsco.com does not want to maintain user identities for all employees at smcompany.com, but access to sensitive portions of the Partsco.com Web site must be controlled. To do this, partsco.com maintains a limited number of profile identities for users at smcompany.com. One profile identity is maintained for engineers and one profile identity is maintained for buyers.

When an employee of smcompany.com accesses partsco.com, user attributes are sent in a secure manner from smcompany.com to partsco.com, which uses them to determine what profile identity should be used to control access.

More information:

Solution 2: SSO Using User Attribute Profiles (see page 53)
Use Case 3: Single Sign-on with No Local User Account

In Use Case 3, smcompany.com offers employee discounts by establishing a partnership with discounts.com.

An employee of smcompany.com authenticates at an employee portal at www.smcompany.com and clicks a link to access discounts at discounts.com. The employee is taken to discounts.com's web site and presented with the discounts available for smcompany.com employees, without having to sign on to discounts.com's Web site.

The following illustration shows this use case.

Discounts.com does not maintain any identities for employees of smcompany.com—the company allows all employees of smcompany.com to access discounts.com as long as they have been authenticated at smcompany.com. When an employee of smcompany.com accesses discounts.com, authentication information is sent in a secure manner from smcompany.com to discounts.com. This information is used to allow access to discounts.com.

Additional attributes, such as user name are passed from smcompany.com to discounts.com to personalize the interface for the individual user.

More information:

Solution 3: SSO with No Local User Account (see page 55)
Use Case 4: Extended Networks

In Use Case 4, smcompany.com, ahealthco.com, and discounts.com all participate in an extended federated network. This case is an extension of the use cases presented previously.

In this network, not all of ahealthco.com’s customers work at smcompany.com, so ahealthco.com provides discounts to its customers by establishing a relationship between themselves and discounts.com. Since ahealthco.com maintains user identities for every customer, it is possible for ahealthco.com to manage local credentials, such as a password for each user. By managing local credentials, ahealthco.com can authenticate users and provide single sign-on access to its partners.

In this extended network, the users access each Web site differently:

- User1 accesses health benefit information at ahealthco.com’s web site. User 1 may also choose to access partsco.com’s Web site by clicking on the PartsSupplier link at smcompany.com, her employee portal. She can also click a link at her employee portal to access discounts at discounts.com.
User2 authenticates at ahealthco.com's web site and clicks a link to access discounts at discounts.com, without having to sign on to discounts.com's web site. The discounts presented to User2 reflect the business arrangement between ahealthco.com and discounts.com. Because User2 is an employee of smcompany.com, he can also click a link at ahealthco.com and access the employee portal at smcompany.com without having to sign on to smcompany.com's web site.

User3 (not shown in example), is a customer of ahealthco.com, but is not an employee of smcompany.com. User3 authenticates at ahealthco.com's web site and clicks a link to access discounts at discounts.com without having to sign on to discounts.com's web site. The discounts presented to User3 reflect the business arrangement between ahealthco.com and discounts.com. Since User3 is not an employee of smcompany.com, User3 cannot access smcompany.com's web site.

More information:

Solution 4: SSO in an Extended Network (see page 57)

**SPS Roles in a SiteMinder Federated Environment**

The SPS can provide solutions to federation use cases in one of two roles:

- As a standard proxy server that replaces the SiteMinder Web Agent
- As a federation gateway

The primary distinction between these two roles is the configuration and deployment effort required. The proxy server that replaces the Web Agent still requires that you set up a separate server and servlet engine to run the Federation Web Services application.

The proxy server acting as a federation gateway has the components of the Web Agent and the Federation Web Services application built-in. A dedicated server and servlet engine are not configured separately, which simplifies the federation setup.

**Solutions for SPS Use Cases**

The following sections show SPS solutions to the federation use cases.

**Solution 1: SSO Based on Account Linking**

Solution 1 illustrates how Federation Security Services can be deployed at smcompany.com and ahealthco.com to solve Use Case 1 (see page 46): Single Sign-on Based on Account Linking.
The following figure shows the solution based on account linking. SiteMinder v6.x is deployed at both sites and the installations are the same for both smcompany.com and ahealthco.com. The SPS with the Web Agent Option Pack or the SPS federation gateway can be installed on the Web server system and the Policy Server with the Policy Server Option Pack are installed on another machine.

The FWS application at the producing side provides the service that retrieves assertions. The FWS application at the consuming side provides the service that consumes assertions.
Using SAML 1.x Artifact Authentication for Solution 1

The process that follows is one solution for single sign-on with account linking. This solution uses the SAML 1.x artifact profile. There are other solutions for this use case that involve other profiles (SAML 1.x POST and SAML 2.0 Artifact and POST). For these solutions, see the CA SiteMinder Federation Security Services Guide.

In this solution, smcompany.com is acting as the producer site. When an employee of smcompany.com accesses an employee portal at www.smcompany.com, the sequence of events is as follows:

1. The SPS provides the initial authentication.
2. When the employee clicks a link at smcompany.com to view her health benefits at ahealthco.com, the link makes a request to the Intersite Transfer Service at www.smcompany.com.
3. The Intersite Transfer Service calls the assertion generator, which creates a SAML assertion, inserts the assertion into the SiteMinder session server, and returns a SAML artifact.
4. The SPS redirects the user to www.ahealthco.com with the SAML artifact, in accordance with the SAML browser artifact protocol.

Ahealthco.com is acting as the consumer site. The redirect request with the SAML artifact is handled by the SAML credential collector Federation Web Services at ahealthco.com.

The sequence of events is as follows:

1. The SAML credential collector calls the SAML artifact authentication scheme to obtain the location of the assertion retrieval service at smcompany.com.
2. The SAML credential collector calls the assertion retrieval service at www.smcompany.com.
3. The assertion retrieval service at www.smcompany.com retrieves the assertion from the SiteMinder session server and returns it to the SAML credential collector at ahealthco.com.
4. The SAML credential collector then passes the assertion to the SAML artifact authentication scheme for validation and session creation and proceeds to issue a SiteMinder session cookie to the user’s browser.
5. At this point the user is allowed access to resources at ahealthco.com based on policies defined at the Policy Server at ahealthco.com and enforced by the SPS at ahealthco.com.

In this example, the administrator at smcompany.com uses the Policy Server User Interface to configure an affiliate for ahealthco.com. The affiliate is configured with an attribute that is a unique ID for the user. This causes the assertion generator to include that attribute as part of the user profile in a SAML assertion created for ahealthco.com.
The administrator at ahealthco.com uses the Policy Server User Interface to configure a SAML artifact authentication scheme for smcompany.com. The authentication scheme specifies the location of the assertion retriever service at smcompany.com, how to extract the unique user ID from the SAML assertion, and how to search the user directory at ahealthco.com for the user record that matches the value extracted from the assertion.

**Solution 2: SSO Using User Attribute Profiles**

Solution 2 shows how SiteMinder Federation Security Services can be deployed at smcompany.com and partsco.com to solve Use Case 2 (see page 47): Single Sign-on Based on User Attribute Profiles.

SiteMinder v6.x is deployed at both sites. The interactions between the user and each site is similar, where partsco.com is acting as the consuming authority. The FWS application at the producing side provides the service that retrieves assertions. The FWS application at the consuming side provides the service that consumes assertions.

The following illustration is similar for SAML 1.x, SAML 2.0, and WS-Federation; however, the Federation Web Services components are different as follows:

- For SAML 1.x, the Assertion Retrieval Service (for artifact profile only) is at the Producer and the SAML credential collector is at the SP.
- For SAML 2.0, the Artifact Resolution Service (for artifact binding only) is at the IdP and the Assertion Consumer Service at the SP.
- For WS-Federation, the Single Sign-on Service is at the AP and the Security Token Consumer Service is at the RP.

**Note:** WS-Federation only supports HTTP-POST binding.
The configuration is similar to Solution 1: Single Sign-on based on Account Linking, except for the following:

- The administrator at smcompany.com defines the consumer/SP for partsco.com with an attribute specifying the user’s department at the company. The assertion generator will include this attribute as part of the user profile in the SAML assertion created for partsco.com.

- The administrator at partsco.com defines an authentication scheme (artifact, post, or WS-federation) for smcompany.com. The scheme extracts the department attribute from the SAML assertion and searches the user directory at partsco.com for the user record that matches the department value taken from the assertion. The administrator defines one user profile record for each department that is allowed to access partsco.com's web site.

**Solution 3: SSO with No Local User Account**

Solution 3 shows how SiteMinder Federation Security Services can be deployed at smcompany.com and discounts.com to solve Use Case 3 (see page 48): Single Sign-on with No Local User Account.

SiteMinder v6.x is deployed at smcompany.com by installing the SPS on one machine, the Web Agent Option Pack on another machine and installing the Policy Server with the Policy Server Option Pack on a third machine. The SAML Affiliate Agent is installed at discounts.com. It only supports SAML 1.0.

The FWS application at the producing side provides the assertion retrieval service. The FWS application at the consumer side provides the SAML credential collector.

**Note:** The SPS federation gateway does not support SAML 1.0 and therefore cannot act as a producer for the SAML Affiliate Agent.
The following figure shows single sign-on with no local user account.

Smcompany.com is acting as a SAML 1.x producer. When an employee of smcompany.com accesses an employee portal at www.smcompany.com, the following occurs:

1. The SPS provides the initial authentication.
2. When the employee clicks a link at www.smcompany.com to access deals at discounts.com, the link makes a request to the SPS at www.smcompany.com.
3. The SPS at www.smcompany.com calls the assertion generator, which creates a SAML assertion, inserts the assertion into the SiteMinder session server, and returns a SAML artifact.
4. The SPS redirects the user to www.discounts.com with the SAML artifact in accordance with the SAML browser artifact protocol.

Discounts.com is acting as the consumer site. The redirect request with the SAML artifact is handled by the SAML Affiliate Agent at www.discounts.com, as follows:

1. The SAML Affiliate Agent obtains the location of the assertion retrieval service at www.smcompany.com from a configuration file.
2. The SAML Affiliate Agent calls the assertion retrieval service at www.smcompany.com.
3. The assertion retrieval service at www.smcompany.com retrieves the assertion from the SiteMinder session server and returns it to the SAML affiliate agent at www.discounts.com.

4. The SAML Affiliate Agent then validates the SAML assertion and issues a SiteMinder affiliate session cookie to the user’s browser.

5. The user is allowed access to resources at discounts.com.

The administrator at smcompany.com uses the Policy Server User Interface to configure an affiliate for discounts.com. The affiliate is configured with attribute information to be passed to discounts.com. The assertion generator will include those attributes as part of the user profile in a SAML assertion created for discounts.com.

The administrator at discounts.com configures the SAML Affiliate Agent with information about the discounts.com site, the location of the assertion retriever service at smcompany.com, and what resources are to be protected by the affiliate defined at smcompany.com.

**Solution 4: SSO in an Extended Network**

Solution 4 illustrates how SiteMinder Federation Security Services can be deployed at smcompany.com, ahealthco.com, and discounts.com to solve Use Case 4 (see page 49): Extended Networks.
The following illustration shows an extended network. SAML 1.x is the protocol being used.

SiteMinder is deployed at smcompany.com and ahealthco.com. At smcompany.com, the SPS with the Web Agent Option Pack can be installed across two machines or the SPS federation gateway can be installed on one machine. The Policy Server with the Policy Server Option Pack is installed on another machine. At ahealthco.com, the SPS with the Web Agent Option Pack can be installed across two machines and the Policy Server with the Policy Server Option Pack is installed on another machine. At discounts.com, the SAML Affiliate Agent is installed.

The FWS application at the producing side provides the service that retrieves assertions. The FWS application at the consuming side provides the service that consumes assertions.
In Solution 4:

- smcompany.com acts as a producer for User1 and a consumer for User2
- ahealthco.com acts as a consumer for User1 and a producer for User2 and a producer for User3
- discounts.com acts as a consumer for User1, User2, and User3

The administrator for smcompany.com has configured two entities in an affiliate domain, which represents ahealthco.com and discounts.com. These sites are configured in a similar manner as in Examples 1 and 3 described previously, but the configurations have been extended as follows:

- At smcompany.com, the administrator has configured a SAML authentication scheme (artifact or POST). For User2, the authentication scheme enables smcompany.com to act as a consumer for ahealthco.com.
- At ahealthco.com:
  - The administrator has configured an affiliate object that represents smcompany.com so an assertion is produced for User2. This makes single sign-on to smcompany.com possible.
  - The administrator has configured an affiliate object that represents discounts.com so an assertion is produced for User2 and User3. This makes single sign-on to discounts.com possible.
- At discounts.com, the administrator has configured the SAML Affiliate Agent to act as a consumer for smcompany.com, as in Example 3 (an arrow connecting the two sites is not shown in the illustration). The administrator at discounts.com has also added configuration information about ahealthco.com so that the SAML Affiliate Agent can consume assertions from ahealthco.com for User2 and User3.

**Cookieless Federation**

Certain devices or environments cannot use cookies to establish user session and provide single sign-on.

One type of session scheme you can use in a federated environment is a cookieless scheme. The cookieless federation scheme is used to establish single sign-on. Verify that FWS-generated cookies (session and attribute) are not sent back to clients using mobile devices that do not support cookies.
Cookieless Federation at the Producing Site

At the site producing assertions, the process for a cookieless transaction is as follows:

1. The SPS verifies if cookieless federation is enabled for the virtual host requesting the redirect.
2. The SPS verifies if the session scheme is a rewritable scheme, such as the simple_url scheme.
3. If the scheme is rewritable, SPS determines whether a session key has been created for the session and if this key is available to use.
4. SPS checks to see if the Location header in the HTTP response meets one of the following conditions:
   - It is being rewritten.
   - It is the same as the host of the request.
5. SPS rewrites the redirect response to include the session key information in the redirected URL.

Cookieless Federation at the Consuming Site

At the site consuming assertions, if cookieless federation is enabled, the SPS replacing the Web Agent processes redirects using SAML authentication at the backend server.

In a cookieless federation, the SPS processes the request as follows:

1. The SPS receives a request from cookieless device, such as a mobile phone.
2. The SPS verifies if the cookieless federation is enabled for the virtual host requesting the redirect.
3. SPS then checks to see if the following conditions have been met:
   - The response from the backend server is a redirect.
   - The response contains an SMSESSION cookie.
   If these two conditions are met at the same time, it indicates that a SAML authentication has occurred at the backend server from the FWS application.
4. The SPS retrieves the session scheme being used.
5. The SPS creates an associated cookieless session and adds the session information to its session store.
6. If the session scheme is rewritable, such as a simple URL session scheme, the SPS rewrites the location header with the session key.
7. If the SPS determines that a cookieless federated session conversion has occurred, the SPS deletes the SMSESSION cookie from the response going to the browser.

8. The SPS then checks to see if attribute cookies should also be deleted. It does this by checking the deleteallcookiesforfed parameter (see page 104). If this parameter is set to yes, SPS deletes all the other cookies from the response going to the browser.

**Enable Cookieless Federation at the Consuming Side**

When the SPS replaces the Web Agent at the side consuming assertions, the cookieless federation parameters are enabled for any cookieless session scheme implemented by the SPS.

**To enable cookieless federation for SPS at the consuming side**

1. Open noodle.properties file from \(\textit{sps\_home/secure-proxy/Tomcat/properties}\).

2. Remove the '#' from the following two lines, and save the file.
   - \(\text{filter\_cookielessfederation\_class=org.tigris.noodle.filters.CookielessFedFilter}\)
   - \(\text{filter\_cookielessfederation\_order=1}\)
   
   The settings are saved.

3. Open the server.conf file located at \(\textit{sps\_home/secure-proxy/proxy-engine/conf}\).

4. Add the following code to the virtual host section for the virtual host that is serving the FWS:
   
   ```
   cookielessfederation="yes"
   ```

5. Save the file.

   The SPS is configured for cookieless federation at the consuming partner.

**SPS as a Web Agent Replacement**

To provide federated single sign-on, the SPS may be used as a substitute for the SiteMinder Web Agent. The SPS, and the Web Agent Option Pack combine to provide the Federation Web Services (FWS) application, which is a collection of servlets packaged as a Web application. This application provides much of the SiteMinder federation functionality.

Knowledge of SiteMinder Federation Security Services is required for anyone configuring SPS in a federated environment. For more information on Federation Security Services, see the CA SiteMinder Federation Security Services Guide.
The following figure shows an environment where the SPS replaces a SiteMinder Web Agent.

**Important!** If you choose to use the SPS in place of the Web Agent for a federated environment, the Web Agent Option Pack requires a dedicated web server and servlet engine separate from the web server and servlet engine included in the SPS.

**Prerequisites for Using the SPS as a Web Agent Replacement**

Before you configure SPS for use in a SiteMinder Federation Security Services environment, consider the following:

- The SiteMinder environment must be configured according to the information in the *CA SiteMinder Federation Security Services Guide*. We recommend that you configure a Federation environment with a standard Web Agent to confirm that Federation Security Services is configured properly.

- After you confirm that the federation environment is working properly, install the Web Agent Option Pack on the SPS system, or on a separate system.

- Install a Servlet Engine for use by the Web Agent Option Pack.

For more information about FSS and servlet engines, see the *CA SiteMinder Federation Security Services Guide*.

- In the SiteMinder Policy Server User Interface, define the host information (server and port number) for the SPS system that generates assertions. The SPS host is defined in the Server field of the appropriate properties dialog for the federated partner you are specifying.
Configuring the SPS as a Web Agent Replacement for Federation

The configuration process for the SPS to operate in a federated environment is similar to the standard SPS configuration process.

The overall configuration process for the SPS federation gateway is as follows:

1. Install the SPS.
2. Run the configuration wizard.
3. Specify the general server settings in the server.conf file. Though there are defaults for most of the server.conf settings, you can modify such settings as logging, session schemes, or virtual host settings.
4. Define proxy rules in the proxyrules.xml file so that requests are directed to the backend servers.
   - At the enterprise producing assertions, define a proxy rule that forwards requests to the backend server hosting FWS. At the side consuming assertions, there must be a rule that forwards requests to the destination server after the user is permitted access to the target resource.
5. (Optional) If you want to configure virtual hosts for the SPS, you can modify the Apache web server file (httpd.conf), for example,

More information:
- Configuring the Apache Web Server (see page 71)
- Configuring the SPS Server Settings (see page 73)
- Configuring Proxy Rules (see page 117)

SPS as a Federation Gateway

The SPS federation gateway simplifies the configuration involved in a federated environment. Typically, you have a federated environment where partners are communicating through many web servers. Each web server requires that you install and configure the Web Agent and the Web Agent Option Pack.

If you enable the SPS as a federation gateway, the number of components that you have to install and set-up is reduced. The SPS federation gateway has the standard embedded components of the SPS and the Federation Web Services application provided by the Web Agent Option Pack.

Note: Knowledge of SiteMinder Federation Security Services is required for anyone configuring SPS in a federated environment. For more information about Federation Security Services, see the CA SiteMinder Federation Security Services Guide.
The following illustration shows the difference with or without the SPS federation gateway.

**Federated Environment without the SPS Federation Gateway**

![Diagram of Federated Environment without the SPS Federation Gateway]

**Federated Environment with the SPS Federation Gateway**

![Diagram of Federated Environment with the SPS Federation Gateway]
Prerequisites for Using the Federation Gateway

Before you set up the SPS as a federation gateway, consider the following:

■ The SiteMinder environment must be configured according to the information in the CA SiteMinder Federation Security Services Guide. Verify that the Policy Server side components for federation are configured.

■ Install the SPS and enable the enablefederationgateway setting when prompted.

■ In the SiteMinder Policy Server User Interface, be sure to define the host information (server and port number) for the SPS system that generate assertions. The SPS host is defined in the Server field of the appropriate properties dialog for the federated partner you are specifying.

Configuring the SPS Federation Gateway

The SPS federation gateway can sit at the producer site and consumer site.

The overall configuration process for the SPS federation gateway is as follows:

1. Install the SPS.

2. Run the configuration wizard.

3. Specify the general server settings in the server.conf file. Though there are defaults for most of the server.conf settings, you may want to modify such settings as logging, session schemes, or virtual host settings.

4. Define proxy rules in the proxyrules.xml file so that requests are directed to the backend servers.

At the enterprise producing assertions, federation requests are forwarded to the Tomcat server embedded in the SPS. The Tomcat server hosts the FWS application. Proxy rules and filters have no relevance when the federation request gets processed.

At the enterprise consuming assertions, you need to define a proxy rule that forwards requests to the destination server after the user is permitted access to the target resource.

5. (Optional) You can modify the Apache web server file (httpd.conf).
Limitations of the SPS Federation Gateway

Note the following limitations when using the SPS federation gateway:

- The prefilters and postfilters (both built-in and custom-configured) do not execute when federation resources are being requested. For non-federated requests that are fired for the default context, these filters execute as usual.

- Proxy rules do not execute when federated resources are being requested. For non-federated requests that are fired for the default context, these rules execute as usual.
Overview Single Sign-on Security Zones

SSO security zones provide configurable trust relationships between groups of applications within the same cookie domain. Users have single sign-on within the same zone, but can be challenged when entering a different zone, depending on the trust relationship defined between the zones. Zones included in a trusted relationship do not challenge a user that has a valid session in any zone in the group.

SiteMinder Web Agents implement single sign-on security zones. Each zone must reside on a separate Web Agent instance. All Web Agents configured through the same agent configuration object belong to the same single sign-on zone.

Cookies generated by the Web Agent identity security zones. 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 generates session cookies and identity cookies with unique names so that the zone affiliation is reflected in the cookie names.

Note: For detailed information about SSO security zones, see the CA SiteMinder Web Agent Guide.

Parameters for Security Zones

The two single sign-on parameters listed following are manually added to the Web Agent configuration objects in the policy store. These settings can also be used in local configuration files and are added to the sample local configuration files laid down during installation.
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. When this parameter is not empty, SiteMinder generates cookies using the following convention: ZonenameCookiename. The default is empty and uses SM as a zone name, which gives the cookies the following default names:

- SMSESSION
- SMIDENTITY
- SMDATA
- SMTRYNO
- SMCHALLENGE
- SMONDENIEDREDIR

Example: Setting the value to Z1 creates the following cookies:

- Z1SESSION
- Z1IDENTITY
- Z1DATA
- Z1TRYNO
- Z1CHALLENGE
- Z1ONDENIEDREDIR

SSOTrustZone

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. The default is empty, or can be SM or the SSOZoneName if provided.
Configure SPS Security Zones

SSO security zones are intended for situations where SiteMinder administrators want to segment the 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 security zones do not exist:

1. The user accesses an application (APP1). SiteMinder challenges the user for credentials. The user logs into SiteMinder, and creates an SMSESSION cookie.
2. The user accesses a second application (APP2), and SiteMinder challenges the user again. (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 an SMSESSION cookie overwriting the old one with the new logged in session for APP2.
3. The user returns to APP1 and is challenged again, because the user lost the original APP1 session, and the APP2 session possibly has not been accepted for APP1. Therefore, SSO does not occur between APP1 and APP2.

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 applications.

To configure SPS Security Zones

1. Install a Secure Proxy Server and configure it with the Policy Server. Consider all resources protected by this SPS as belonging to the same security zone. By default, users is not challenged when accessing multiple resources in this zone in the same session.
2. In the local agent configuration file for this SPS, add a value to the parameter SSOZoneName
3. On another Web Agent within the same cookie domain, configure the following two parameters:
   - SSOZoneName
   - SSOTrustedZone
Example

Configure SSOZoneName=A in Zone1.

Configure SSOZoneName=B and SSOTrustedZone=A in Zone2.

Users who are authenticated in Zone B and have previously been authenticated in Zone A can continue to access resources in Zone A in the same session without being rechallenged.

Note: For information about configuring security zones beyond this basic use case, see the Web Agent Configuration Guide.
Chapter 7: Configuring the Apache Web Server

This section contains the following topics:

Apache Web Server Configuration File (see page 71)

Apache Web Server Configuration File

The SPS proxy engine works with an embedded Apache web server. If, for example, you want to configure virtual hosts for the SPS, you can modify the Apache web server configuration.

The configuration file for the Apache web server is the httpd.conf file, which is located in:

\texttt{sps\_home/secure-proxy/httpd/conf/}

\textbf{Important!} If you change any Apache setting while upgrading the SPS, or during any other reconfiguration scenario, restart the SPS services to reflect the changes. In addition, restart the SPS with the new settings (for example, the new port number).
The SPS is configured through settings contained in the server.conf file. These settings in the file are groups of name/value pairs or directives that the SPS reads at startup.

After the SPS is operating, it verifies the values in this file to determine if any changes have been made to the SPS Web Agent log level settings. If changes are detected, the affected settings are reloaded so that the SPS can be dynamically updated without interrupting network traffic.

The server.conf file is located in the following directory:

`sps_home/secure-proxy/proxy-engine/conf`

The file contents are grouped into the following sections:

- **Server**—Includes settings for the server operation, federation gateway operation, logging, and SSL.
- **Session Store**—Defines the session store.
- **Service Dispatcher**—Defines setting for this global server parameter.
- **Proxy and Redirect Services**—Specifies the connection pools and filters for the proxy service and the class for the redirect services.
- **Session schemes**—Defines the session schemes.
- **User agents**—Specifies types of user agents.
- **Virtual Hosts**—Identifies the default virtual host and its settings.

Each section is an XML-like element tag. The name of the section is the beginning tag of the XML element and the section ends with a corresponding ending tag. The directives contained in each section follow the format name=value.

Any lines beginning with the # symbol are comments, and are not read when the SPS loads configuration settings.

**Note:** Pathnames on Windows systems use double backslashes (\\), such as `\\logs\\server.log`
Modifying the server.conf File

The settings for the SPS are maintained in the server.conf file located in the following directory:

`sps_home/secure-proxy/proxy-engine/conf`

To change the settings in the server.conf file

1. Open the file in a text editor.
2. Edit the directives, as necessary.
3. Restart the SPS.

The settings are changed.

General Server Settings in the server.conf File

The `<Server>` section of the server.conf file contains parameters for server connectors, Federation, logging, and SSL. These parameters are described in the sections that follow.

HTTP Connection Parameters

#Define the listeners between #HTTP listener and proxy engine.

`worker.ajp13.port=8009`
`worker.ajp13.host=localhost`
`worker.ajp13.reply_timeout=0`
`worker.ajp13.retries=2`

**Note:** The values for the connector directives are not contained in quotes. Values for other types of directives are contained in quotes.

The name/value pairs are:

`worker.ajp13.port=8009`

   Specifies the port for the ajp13 connector.

`worker.ajp13.host=localhost`

   Specifies the local ajp13 host as the local host.
Additional tuning parameters can be defined for the connection between the HTTP listener and the proxy engine, including:

**worker.ajp13.reply_timeout**

Specified the maximum time in milliseconds that can elapse between any two packets received from the proxy engine after which the connection between HTTP listener and the proxy engine is dropped. A value of zero makes it wait indefinitely, until a response is received.

*Default: 0*

**worker.ajp13.retries**

Specifies the maximum number of times that the worker sends a request to the proxy engine in a communication error.

*Default: 2*

---

**Tomcat Tuning Parameters in the server.conf File**

A Tomcat server is embedded in the SPS. The Tomcat server provides a servlet container and servlet engine.

The following excerpt is from the Tomcat tuning section in the server.conf file:

```text
#Define AJP13 tuning parameters
#Number of request waiting in queue (queue length)
#Number of threads created at initialization time
#Maximum number of concurrent connections possible
worker.ajp13.accept_count=10
worker.ajp13.min_spare_threads=10
worker.ajp13.max_threads=100
worker.ajp13.connection_pool_timeout=0
worker.ajp13.max_packet_size=8192
```
The Tomcat tuning directives are listed following.

**worker.ajp13.accept_count**

Defines the number of requests waiting in the queue when all possible request processing threads are in use. Any requests received when the queue is filled are refused.

**Default:** 10

**worker.ajp13.min_spare_threads**

Defines the minimum number of idle threads at any time, waiting for new requests to arrive. min_spare_threads must be greater than 0.

**Default:** 10

**worker.ajp13.max_threads**

Defines the maximum number of concurrent connections possible, the pool will not create more than this number of threads.

**Default:** 100

**worker.ajp13.connection_pool_timeout**

Defines the maximum time (in seconds) that the idle connections (between apache and tomcat over mod-jk) remain in the connection pool before timing out. The default is zero, which means that connections never timeout.

**Default:** 0

**worker.ajp13.max_packet_size**

Defines the maximum packet size in Bytes. The maximum value is 65536

**Default:** 8192

---

**Resolve Differences in Cookie Specification for Different Version of Tomcat**

Tomcat version 5.5 changed its behavior for handling cookies. Tomcat version 5.5 by default puts quotes around a cookie. Previous versions of Tomcat did not put quotes around a cookie. Tomcat sends the cookie back to the browser. SPS r12.0 SP 3 uses Tomcat version 5.5. If your deployment requires visiting an earlier version of SPS, the cookie cannot be decoded. The earlier version of SPS is using Tomcat version 5.0, which does not put quotes around the cookie.

To ensure that the cookie behavior is compatible between different versions of SPS, set the addquotestobrowsercookie parameter in the server.conf file to "no". The Tomcat org.apache.catalina.STRICT_SERVLET_COMPLIANCE variable is set to "TRUE". Tomcat parses the cookie according to the servlet specification, which means that no quotes are added. When the addquotestobrowsercookie parameter is set to "yes", the SPS enables the default Tomcat version 5.5 cookie behavior.
Parsing the Equal Sign in a Cookie

Tomcat 5.5 and later adds an equals (=) sign to the cookie. The SPS allows this practice and parses cookie values that contain an equal sign. The default value for the allowequalsincookievalue parameter in the server.conf file is "yes".

Set the allowequalsincookievalue parameter to "no" if you want parsing of the cookie value to terminate when the parser encounters an equal sign.

Federation Settings in the server.conf File

The federation settings in the server.conf file enable the SPS to act as a federation gateway within a SiteMinder federation network.

The code excerpt that follows is the <federation> section on the server.conf file:

```xml
# Provide the values for the Federation related parameters here
#
# enablefederationgateway - "yes" or "no" - Enable or Disable SPS Federation Gateway
# fedrootcontext - Name of the Federation root context ("affwebservices" by default)
# authurlcontext - Path of the Authentication URL (without the jsp file name)
# (siteminderagent/redirectjsp by default)
# protectedbackchannelservices - Names of protected Backchannel services

<federation>
  enablefederationgateway="yes"
  fedrootcontext="affwebservices"
  authurlcontext="siteminderagent/redirectjsp"
  protectedbackchannelservices="saml2artifactresolution,saml2certartifactresolution,saml2attributeservice,saml2certattributeservice,assertionretriever,certassertionretriever"
</federation>
```

The federation parameters are as follows:

**enablefederationgateway**

Enables the SPS to act as a federation gateway proxy server.

**Limits:** yes or no

This parameter is set during the installation.

**fedrootcontext**

Specifies the root context of the federation web services application. Do not change this parameter.

**Default:** affwebservices
authurlcontext

Specifies the alias to the redirect.jsp file. When a user requests a protected federation resource and they do not have a SiteMinder session at the site that produces assertions, the user is sent to this URL which points to a redirect.jsp file. The user is redirected to the Web Agent at the producing site where they are presented with an authentication challenge and upon successfully logging in, establish a session.

Default: siteminderagent/redirectjsp.

protectedbackchannelservices

Lists the services that require a secure back channel for communication.

More information:

Using the SPS with Federation Security Services (see page 45)

Logging Settings in the server.conf File

The <Server> element allows you to specify the logging settings for the SPS. The logging defined in this section focuses on the startup and shutdown of the SPS. The logging configuration is done in the Web Agent configuration file (webagent.conf or localconfig.conf) or the Agent Configuration Object configured at the Policy Server.

The logging section has the following format:

```plaintext
# Logging for the server
# 1 - FATAL
# 2 - ERROR
# 3 - INFO
# 4 - DEBUG
loglevel=3
logconsole=yes
logfile=yes
logappend=no

# Note: If logfilename is specified as a relative file, it will be relative to proxy-engine/
logfilename=logs/server.log
```
loglevel

Sets the log level of the SPS server log. The higher the log level, the greater the detail of information that is recorded in the SPS log.

The log levels are as follows:

1

Indicates the least amount of detail in the log. Only fatal errors are recorded at log level 1.

2

Reports any error messages. Any errors that occur during processing are recorded at log level 2.

3

Indicates that warnings and other informational messages are recorded in the log.

4

Indicates debugging.

logconsole

Specifies that the log file is written to the console window.

logfile

Specifies that the log information is written to a file. Set the parameter to yes to write the file to the location specified in the logfilename parameter. Set it to no if you do not want to write the log to a file.

logappend

Indicates that the log information is appended to a log file when the SPS starts. Set this parameter to yes to append data to an existing log file when the SPS restarts. Set this parameter to no if you do not want to append data.

logfilename

Defines the path and filename of the SPS log file. The `\logs\` portion indicates that the logs appear in the default location `sps_home\proxy-engine\logs`. 
Modify ServerPath in WebAgent.conf for Logging

If you configure a Web Agent for virtual hosts, each host must have its own Web Agent cache, log file, and health monitoring resources. To help ensure that resources are unique, configure the ServerPath parameter.

The ServerPath parameter specifies a unique identifier for the Web Agent resources of cache, logging, and health monitoring. For each server instance to have its own set of these Agent resources, the value of the ServerPath parameter must be unique.

For example, you can set the ServerPath parameter to the directory where the web server log file is stored, such as `server_instance_root/logs`.

If you have virtual hosts in your environment, verify that the ServerPath parameter is in each WebAgent.conf file.

**To verify that the ServerPath parameter is in each WebAgent.conf file**

1. Navigate to the WebAgent.conf file in the directory `sps_home\secure-proxy\proxy-engine\conf\defaultagent`
2. Open the file.
3. Check that the ServerPath setting is configured to a unique string or path.
   - For Windows, you can specify any unique string.
   - For UNIX, specify a unique system path.
4. Save the WebAgent.conf file.

HttpClient Logging

You can enable HttpClient logging by setting the `httpclientlog` parameter to "yes". This parameter is located in the `<Server>` section of the `server.conf` file. By default, this parameter is set to "no".

We recommend that you enable HttpClient logging only for debugging. In a production environment, enabling logging can cause performance degradation.

Configure HttpClient Logging

You can configure various aspects of HttpClient logging by setting values to parameters in the `httpclientlogging.properties` file. This file is located in the `sps_home\Tomcat\properties` directory.

**Important!** Because of potential performance degradation, do not enable HttpClient logging in a production environment.
The httpclientlogging.properties file has the following configurable parameters:

**java.util.logging.FileHandler.formatter**

Description: Specifies the name of the formatter class

Limits:

- java.util.logging.SimpleFormatter — writes brief summaries of log records
- java.util.logging.XMLFormatter — writes detailed descriptions in XML format

Default: java.util.logging.SimpleFormatter

**java.util.logging.FileHandler.pattern**

Description: Specifies the name of the HttpClient log file.

Limits:

- sps_home\proxy-engine\logs\httpclient%g.log

%g represents the generation number of the rotated log file.

**java.util.logging.FileHandler.count**

Description: Specifies the number of output files in a cycle

Default: 10

**java.util.logging.FileHandler.limit**

Description: Specifies an approximate maximum number of bytes to write to any on log file.

Limits: If set to zero, there is no limit.

Default: 5,000,000

In addition, you can specify the content of the logs with the following parameters:

**Note:** The value is always FINEST.

- org.apache.commons.httpclient.level=FINEST
  
  Specifies context logging only

- httpclient.wire.header.level=FINEST
  
  Specifies header wire and context logging

- httpclient.wire.level=FINEST
  
  Specifies full wire (header and content) and context logging
SSL Settings in the server.conf File

The <sslparams> section in the server.conf file contains the settings required to enable Secure Sockets Layer (SSL) communications between the SPS and destination servers.

The SSL configuration section is listed following.

```xml
<sslparams>
    # Set the SSL protocol version to support: SSLv3, TLSv1
    # WARNING: SSL version 2 is no longer supported.
    versions="SSLv3"

    ciphers="-RSA With Null SHA,+RSA With Null MD5,-RSA With RC4 SHA,+RSA With RC4 MD5,+RSA With RC2_CBC_MD5,+RSA With DES_CBC_SHA,+RSA With DES_CBC_MD5,+RSA With 3DES_EDE_CBC_SHA,+RSA With 3DES_EDE_CBC_MD5,+RSA Export With RC4 40 MD5,-RSA Export With DES 40 CBC SHA,+RSA Export With RC2 40 CBC MD5,-DH RSA With DES_CBC_SHA,-DH RSA With 3DES_EDE_CBC_SHA,-DH RSA Export With DES 40 CBC SHA,-DH RSA Export With DES 40 CBC SHA,-DH DSS With 3DES_EDE_CBC_SHA,-DH DSS Export With DES 40 CBC SHA,-DH Anon With 3DES_EDE_CBC_SHA,-DH Anon Export With 3DES_EDE_CBC_SHA,-DH Anon Export With DES 40 CBC SHA,-DH Anon Export With DES 40 CBC SHA,-DHE RSA With DES_CBC_SHA,-DHE RSA With 3DES_EDE_CBC_SHA,-DHE RSA Export With DES 40 CBC SHA,-DHE RSA Export With DES 40 CBC SHA,-DHE DSS With DES_CBC_SHA,-DHE DSS With 3DES_EDE_CBC_SHA,A,-DHE DSS Export With DES 40 CBC SHA"

    fipsciphers="+DHE_DSS With_AES_256_CBC_SHA, +DHE_RSA With_AES_256_CBC_SHA, +RSA With_AES_256_CBC_SHA, +DH DSS With_AES_256_CBC_SHA, +DH RSA With_AES_256_CBC_SHA, +DHE_DSS With_AES_128_CBC_SHA, +DHE_RSA With_AES_128_CBC_SHA, +RSA With_AES_128_CBC_SHA, +DH DSS With_AES_128_CBC_SHA, +DH RSA With_AES_128_CBC_SHA, +DHE_DSS With_3DES_EDE_CBC_SHA, +DHE_RSA With_3DES_EDE_CBC_SHA, +RSA With_3DES_EDE_CBC_SHA, +DH DSS With_3DES_EDE_CBC_SHA"

    # Covalent SSL CA certificate bundle and certs path to be converted
    # The bundle and/or certs located at defined location will be converted
    # to binary (DER) format and loaded as SSLParams.
    # NOTE: Only put Base64 (PEM) encoded cert files/bundles in the covalent
    # certificate directory.
    cacertpath="C:\Program Files\CA\secure-proxy\SSL\certs"
    cacertfilename="C:\Program Files\CA\secure-proxy\SSL\certs\ca-bundle.cert"
</sslparams>
The SSL parameters include:

**versions**

Determines the SSL versions supported by the SPS. The entry can be one or more of the following.

- SSLV3
- TLSV1

If you specify more than one version, separate the values by commas.

**ciphers**

Specifies the list of ciphers that can be enabled or disabled. If a cipher is enabled, it is preceded by a + symbol. If a cipher is disabled, it is preceded by a - symbol. If you specify more than one cipher, separate each entry by commas.

**cacertpath**

Specifies the path of the directory that contains the trusted certificate authority information. This path is relative to the install path of the SPS. This value is configured when you run the configuration wizard during the SPS installation; do not change it.

**cacertfilename**

Specifies the fully qualified path name of the file that contains the Certificate Authority bundle of certificates. This file must have a file extension of .cer or .cert, and must be PEM encoded. It must also include the full path to the Certificate Authority (CA) bundle. This value is configured when you run the configuration wizard during the SPS installation.

**maxcachetime**

Specifies the duration, in milliseconds, that the SSL session ID is cached for re-use by the SPS HTTPS client. When a user requests a file via an HTTPS connection, an SSL handshake occurs and an SSL session ID is created. This SSL session ID is used by the SPS and the backend server to identify a user session. When the HTTPS connection is terminated for the user, the SPS caches the SSL session ID for the maximum duration specified by this parameter.

When the same user requests a new HTTPS connection to the backend server, the user can send the SSL session ID that is cached for a faster response. In this case, the SSL session ID provided by the user is compared with the cached SSL session ID. If the SSL session ID is available in cache, the new HTTPS connection is established faster.

**Default:** 120000 milliseconds
General Server Settings in the server.conf File

Setting for Special Characters within the Cookie

The server.conf file includes a parameter, addquotestocookie, for preserving the SPS practice of enclosing the cookie parameter value in double quotes when the value is not zero. You can change the value of addquotestocookie to "no" when you do not want the SPS to add double quotes around the cookie value before sending it to the backend.

The entry in the server.conf file appears as follows:
```
<Server>
  ...
  <sslparams>
    ...
  </sslparams>
  #This parameter is applicable to the cookie added by backend.
  #"yes"--- Default Value. Quotes will be added to the cookie parameter value
  #which contains special characters if the cookie version is other than "0"
  #"no" --- Quotes will not be added to the cookie.
  addquotestocookie="yes"
</Server>
```

Select Character Set for Code Headers

You can specify the character set for the appropriate locale by setting the value of the requestheadercharset parameter. The HttpClient application reads this value to determine how to encode the headers to send to the backend server. Possible values are:

- US-ASCII — Specifies the locale uses US English
- Shift_JIS — Specifies the locale uses Japanese, including support for Japanese usernames.

The default is requestheadercharset="US-ASCII".
Caching POST Data

The following two parameters are included in the server.conf to enable POST data caching and to set the size of cached data:

**enablecachepostdata**
- Description: Specifies whether the SPS caches POST data.
- Limits: Yes, No
- Default: No

**maxcachedpostdata**
- Description: Specifies the size (in KB) of the POST data to cache.
- Limits: None
- Default: 1024 KB

Session Store Settings in the server.conf File

The `<SessionStore>` section of the server.conf file specifies settings for storing user sessions. The session store configuration has the following format:

```xml
<SessionStore>
  # Session Store Information
  class="com.netegrity.proxy.session.SimpleSessionStore"
  max_size="10000"
  clean_up_frequency="60"
</SessionStore>
```

The SessionStore parameters are:

**class**
- Indicates the implementation used to maintain user session. Do not modify this value.
- **Default:** com.netegrity.proxy.session.SimpleSessionStore

**max_size**
- Specifies the maximum size of the session store. The number specified is the maximum number of concurrent sessions in the in-memory session store.
- **Default:** 10000
clean_up_frequency

Sets the interval, in seconds, that the SPS waits before cleaning out expired sessions residing in the session store cache.

**Note:** A long session timeout can decrease the number of session cookies of encrypted and decrypted by the server, but can increase the total number of sessions maintained in cache. If there are users who connect infrequently, specify a shorter cache time and smaller cache size. However, if there are many users who return to your site frequently, use a longer cache time and larger cache size.

### Service Dispatcher Settings in the server.conf File

The `<ServiceDispatcher>` section determines how the SPS provide proxy services. It also specifies the location of the proxy rules XML configuration file.

**Note:** This parameter is a global server configuration parameter and is not configured for each individual virtual host.

The `<ServiceDispatcher>` section is listed following.

```xml
# Service Dispatcher
# This is new since proxy 6.0
# Service Dispatcher is now a global server configuration parameter and is no longer configured on a per virtual host basis.
<ServiceDispatcher>
  class="com.netegrity.proxy.service.SmProxyRules" rules_file="C:\Program Files\CA\secure-proxy\proxy-engine\conf\proxyrules.xml"
</ServiceDispatcher>
```

The parameters in this section are:

**class**

Specifies the service dispatcher used by the SPS to route user requests. Do not change the default value.

**Default:** `com.netegrity.proxy.service.SmProxyRules`

**rules_file**

Specifies the location of the proxyrules.xml file

**Default:** `sps_home/secure-proxy/proxy-engine/conf/proxyrules.xml`
Proxy and Redirect Settings in the server.conf File

The <Service> section of the server.conf file consists of the Proxy Service and the Redirect Service.

The two proxy services predefined for the SPS are:
- forward
- redirect

These services each have a section in the file defined by the <Service name> element. Custom services are similarly defined in the server.conf file, including any parameters set by an administrator.

Proxy Service Configuration

The forwarding service of the SPS forwards requests to the appropriate destination servers according to the conditions and cases in the proxy rules XML configuration file. The parameters for this service are defined in the <Service name="forward"> section of the server.conf file.

Many of the directives manage the connection pool maintained by the SPS. These directives help improve server performance by maintaining connections and alleviating the overhead of establishing a new connection for each request to a destination server.

Additional directives define proxy filters. Proxy filters can be defined here to perform processing tasks before a request is passed to a destination server, and after the destination server returns data to the SPS. Filter names are unique.
The following is an excerpt of the `<Service name="forward">` section.

**Note:** The excerpt does not include most of the comments that you see when you look at the actual server.conf file.

```xml
# Proxy Service
<Service name="forward">
    class="org.tigris.noodle.Noodle"
    protocol.multiple="true"
    http_connection_pool_min_size="4"
    http_connection_pool_max_size="20"
    http_connection_pool_incremental_factor="4"
    http_connection_pool_connection_timeout="1"
    http_connection_pool_wait_timeout="0"
    http_connection_pool_max_attempts="3"
    http_connection_timeout="0"

    # Proxy filters may be defined here to perform pre/post processing tasks.
    # The following format must be used to configure filters:
    # filter.<filter name>.class=<fully qualified filter class name> (required)
    # filter.<filter name>.init-param.<param name1>=<param value1> (optional)
    # filter.<filter name>.init-param.<param name2>=<param value2>
    # filter.<filter name>.init-param.<param name3>=<param value3>

    # The following example illustrates the use of custom filters in a group
    # Defines filter groups with valid Custom filter names.
    #groupfilter.group1="filter1,filter2"
</Service>
```

The parameters in the forward section are:

**class**

Specifies the implementation that provides forwarding services for the SPS. Do not change this value. This value is only exposed to accommodate the rare occasion when a custom service can forward requests specified in the proxy rules XML configuration file.

**Default:** `org.tigris.noodle.Noodle`

**protocol.multiple**

Indicates whether the SPS supports protocols other than HTTP. Specify one of the following values:

**true**

Indicates that protocols other than HTTP are supported. Currently, only HTTPS is supported as an additional protocol in the SPS. True is the default value for this directive.

**false**

Indicates that only the HTTP protocol is supported.
http_connection_pool_min_size

Sets the minimum number of connections to a single destination server that are available for processing user requests.

Default: 4

http_connection_pool_max_size

Sets the maximum number of connections between the SPS and a destination server.

Default: 20

Important! Each connection established by the SPS creates a socket. For UNIX operating systems, if the maximum size of the connection pool is large, you can increase the limit on file descriptors to accommodate the large number of sockets.

http_connection_pool_incremental_factor

Sets the number of connections to a destination server that the SPS opens when all available connections are being used to process requests.

Default: 4

http_connection_pool_connection_timeout_unit

Sets the timeout unit to seconds or minutes.

Default: Minutes

http_connection_pool_connection_timeout

Defines the time, in minutes, the system waits before closing idle connections in the connection pool.

Default: 1

http_connection_pool_wait_timeout

Defines the time, in milliseconds, that the SPS waits for an available connection.

Default: 0

The default, 0, specifies that SPS waits for a connection until notified and invalidates the use of http_connection_pool_max_attempts.
http_connection_pool_max_attempts

Indicates the number of attempts that the system makes to obtain a connection. This directive is only applicable if wait timeout is not zero.

Default: 3

Specify one of the following values:

0

Indicates that the SPS makes attempts indefinitely.

3

Indicates that the SPS makes three attempts.

http_connection_timeout

Defines the time, in milliseconds, spent on host name translation and establishing the connection with the server when creating sockets.

Default: 0; indicates that the system does not enforce a limit.

Note: This timeout explicitly refers to the HTTP connection and not to the connection pool.

filter.filter name.class=fully qualified filter class name

Specifies the filter configured in the server.conf file for each unique filter that is invoked in the proxy rules.

Example: filter.PreProcess.class=SampleFilter

filter.filter name.init-param.param name1=param value1

Specifies the initialization parameters for a filter based on how the filters are defined using the Filter API. Configure the server.conf file to define parameters for each filter.

Example: filter.PreProcess.init-param.param1=value1
**groupfilter.<groupname> = “filtername1,filtername2,……,filtername”**

Specifies the filter groups to implement one or more filters for a given proxy rule. The SPS reads the filter names declared in the group filter and processes the filters in a chain. The groupfilter name can be similarly used as a filter name in proxyrules.xml. When SPS processes a group filter, the pre-filters are processed before post filters even if the order in which they are defined in the groupfilter is reverse.

The following limitations are applicable:

- The filter names must be valid and unique.
- The group filter name must be unique. If you give the same group name for more than one group, only last group survives.
- The group filter name and filter names must be different.

**Example:**

groupfilter.BatchProcess="SampleFilter1, SampleFilter2, SampleFilter3"  

---

**Connection Pooling Recommendations**

Connection pooling is an important part of managing SPS performance. For SPS to provide the best possible service in an enterprise, destination servers must be configured with Keep-Alive messages enabled for connections. Enabling Keep-Alive messages for a destination server allows the SPS to use its connection pooling features.

Keep Alive messages are managed differently for each type of web server.

In addition to enabling Keep-Alive messages, the following settings are recommended for destination servers and the SPS. The table lists the timeout and connection pool recommendations:

<table>
<thead>
<tr>
<th>Settings</th>
<th>HTTP</th>
<th>HTTPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination Server Keep-Alive Maximum Requests (http_connection_pool_max_attempts)</td>
<td>Unlimited</td>
<td>Unlimited</td>
</tr>
<tr>
<td>Destination Server Timeout</td>
<td>Does not time out</td>
<td>Equal to or greater than the HTTP Connection Pool Timeout</td>
</tr>
<tr>
<td>Secure Proxy Server HTTP Connection Pool Timeout Unit (http_connection_pool_connection_timeout_unit)</td>
<td>Set to seconds or minutes; default is minutes.</td>
<td>Set to seconds or minutes; default is minutes.</td>
</tr>
</tbody>
</table>
Proxy and Redirect Settings in the server.conf File

<table>
<thead>
<tr>
<th>Settings</th>
<th>HTTP</th>
<th>HTTPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure Proxy Server HTTP Connection Pool Timeout (http_connection_pool_connection_timeout)</td>
<td>1 minute</td>
<td>1 minute</td>
</tr>
<tr>
<td>Secure Proxy Server HTTP Connection Pool Wait Timeout</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(http_connection_pool_wait_timeout)</td>
<td>0 waits until notified</td>
<td>0 waits until notified</td>
</tr>
<tr>
<td>Secure Proxy Server HTTP Connection Pool Maximum Attempts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(http_connection_pool_max_attempts)</td>
<td>3 value is only useful if</td>
<td>3 value is only useful if</td>
</tr>
<tr>
<td></td>
<td>the HTTP Connection Pool</td>
<td>the HTTP Connection Pool</td>
</tr>
<tr>
<td></td>
<td>Timeout is greater than 0</td>
<td>Timeout is greater than 0</td>
</tr>
<tr>
<td>Secure Proxy Server HTTP Connection Timeout (http_connection_timeout)</td>
<td>0 does not timeout</td>
<td>0 does not timeout</td>
</tr>
</tbody>
</table>

**Redirect Service Configuration**

The redirect service of the SPS sends requests to destination servers. Unlike the forward service, the destination server handles subsequent requests, not the SPS.

The redirect service has the following format:

```xml
<Service name="redirect">
    class=com.netegrity.proxy.service.RedirectService
</Service>
```

The directive is:

- **class**
  - Indicates the implementation that handles redirected requests. This directive must not be modified.
  - **Default**: com.netegrity.proxy.service.RedirectService
Session Scheme Settings in the server.conf File

Session schemes determine how a user’s identity is maintained, providing single sign-on during the course of a session. Each potential session scheme must be included in a SessionScheme section of the server.conf file. Session schemes must be associated with a Java class file that defines the behavior of the session. If no session scheme is specified for a particular type of user agent, the default session scheme is used.

One challenge for enterprise transactions is maintaining user sessions. SiteMinder uses cookies to encapsulate session information. Unlike SiteMinder, the SPS uses several methods and provides a set of APIs to support alternative methods of maintaining sessions that do not rely on cookies. The cookieless session schemes involve some sort of token that reference session information maintained in the SPS in-memory session store. The session store resides in the SPS server’s memory, and can be cleared by restarting the server.

The SPS provides the following out-of-the-box session schemes that you can configure in the server.conf file. These schemes may be associated with user agent types for each virtual host defined in the server.conf file. The association of a session scheme with a user agent type is called a session scheme mapping.

SPS includes the following schemes:

- Default scheme
- SSL ID
- IP Address
- Mini-Cookies
- Simple URL Rewriting
- Device ID

Note: To create additional custom session schemes, you can use the session scheme APIs. If you create your own session schemes using the session scheme API, you must add a <SessionScheme> section to the server.conf file with specific information about the name and Java class associated with your custom session scheme.

More information:

SPS APIs (see page 175)
Establishing a User Session

There are distinct phases for establishing a user session, as follows:

1. Discovery phase
   During this phase of a session, the SPS looks for an appropriate session key based on the user agent type. Session keys are either SiteMinder cookies, or a token that points to the appropriate information in the SPS in-memory session store. As previously discussed, tokens may be in the form of mini-cookies, SSL IDs, device IDs, or other tokens. If no session key can be identified, the Web Agent in the SPS takes over and forwards the request for authentication and authorization and establishes the identity and entitlements of the user.

2. Agent Handling phase
   The SPS contains a Web Agent that communicates with SiteMinder. The Web Agent is responsible for decrypting SiteMinder session information and validating a session with SiteMinder. If a user’s request is accompanied by an SMSESSION cookie, or the SPS has located a user’s session in the session store, the Web Agent validates a user’s request with SiteMinder.

3. Reverse Proxy phase
   In this phase, after the user’s session has been validated, the SPS uses one of its defined services (forward, redirect, or another service) to handle the user’s request. The action of the SPS in this phase is determined by the proxy rules contained in the proxy rules XML configuration file.

   **Note:** For URL rewriting session schemes, content is forwarded to the rewriting mechanism in this phase before being sent back to the user.

Default Session Scheme

The default session scheme is the scheme that the SPS uses to establish and maintain user sessions when no other scheme is specified for a user agent type. The `<SessionScheme>` element contains the name attribute, which is used to identify the session scheme when assigning schemes to user agent types. The server.conf file must contain a default session scheme configuration.

You can configure the default session scheme to use any available session scheme.

The default session scheme section has the following format:

```xml
# Session Schemes
<SessionScheme name="default">
    class="com.netegrity.proxy.session.SessionCookieScheme"
    accepts_smsession_cookies="true"
</SessionScheme>
```
The `<SessionScheme>` element has the following directives:

**class**

Indicates the Java class that contains the default session scheme.

**Default:** com.netegrity.proxy.session.SSLIdSessionScheme

**accepts_smsession_cookies**

Indicates that if a user agent type is associated with the SiteMinder cookies session scheme, users that access resources via that user agent type will maintain session using traditional SiteMinder cookies.

SiteMinder uses cookies to track sessions so a cookies scheme is supported by the SPS. Indicates if SMSESSION cookies are accepted.

Specify one of the following values:

**true**

Indicates that SMSESSION cookies are accepted and used by the session scheme.

**false**

Indicates that SMSESSION cookies are not supported by the session scheme.

### Specifying the Default Session Scheme

The default session scheme is used when no other session scheme is specified for a user agent type.

Default Session Scheme directives are as follows:

**defaultsessionscheme**

Specifies a session scheme other than SiteMinder cookie session scheme as the default scheme. You can modify this entry to include any of your session schemes as the default session scheme.

**Default:** default

**enablewritecookiepath**

Instructs the SPS to rewrite the cookie path from the URI set by the server sitting behind the proxy to the URI of the initial request.

**Default:** no

**enablewritecookiedomain**

Instructs the SPS to rewrite the cookie domain from the domain set by the server sitting behind the proxy to the domain of the initial request.

**Default:** no
SSL ID Session Scheme

A secure sockets layer (SSL) connection includes a unique identifier that is created when an SSL connection is initiated. The SPS can use this unique ID as a token to refer to the session information for a user which is maintained in the SPS in-memory session store. This scheme eliminates cookies as a mechanism for maintaining a user's session.

A limitation of the SSL ID session scheme is that the initial contact with the SPS establishes an SSL session ID. If a user’s SSL session is interrupted, and a new SSL connection is established, the user must be re-authenticated and re-authorized, since the new SSL connection has a connection to a new server, even though it is a virtual server on the same system. This also means that forms used by HTML Forms Authentication Schemes must be served from the same host name as the protected resource.

SSL ID Session Scheme Configuration

The SSL ID section lists the session scheme using the SSL ID.

SSL ID session schemes can be supported without any custom work using the Java classes that are packaged with SPS.

Important! To use the SSL ID authentication scheme, you also have to enable a setting in the Apache Web server's httpd.conf file.

The SSL ID session scheme has the following format:

```
<SessionScheme name="ssl_id">
  class="com.netegrity.proxy.session.SSLIdSessionScheme"
  accepts_smsession_cookies="false"
</SessionScheme>
```

The directives for the ssl_id are as follows:

**class**

Specifies the Java class that handles SSL ID session schemes.

**Default:** com.netegrity.proxy.session.SSLIdSessionScheme

**accepts_smsession_cookies**

Indicates if SMSESSION cookies are accepted. Specify one of the following values:

- **true**
  - Indicates that SMSESSION cookies are accepted and used by the session scheme.

- **false**
  - Indicates that SMSESSION cookies are not supported by the session scheme.
Modifying the httpd.conf File for the SSL ID Scheme

In addition to configuring the SSL ID session scheme in the server.conf file, you have to modify the Apache Web server httpd.conf file to enable SSL.

To modify the httpd.conf file for the SSL ID scheme

1. Open the httpd.conf file located in the directory `sps_home/secure-proxy/httpd/conf`.
2. Locate the line in the file that reads:
   ```
   #SSLOptions +StdEnvVars +ExportCertData +CompatEnvVars
   ```
3. Delete the # symbol from the beginning of the line.
   
   **Note**: For SPS r6.0 SP 3 and later, also remove +CompatEnvVars so that the line reads as follows:
   ```
   SSLOptions +StdEnvVars +ExportCertData
   ```
4. Save the httpd.conf file.
5. Restart the SPS.

IP Address Session Scheme

In environments where IP addresses are fixed, the SPS can use an IP address to refer to a user’s session information in the session store. This scheme eliminates cookies, but may only be used in environments where a user is assigned a fixed IP address.

**More information:**

[Implement a Custom Session Scheme](see page 179)

Mini-cookies Session Scheme

One of the disadvantages of a traditional SiteMinder cookie-based session scheme is the size of the cookies. When the amount of data transferred with each request increases, the cost of access for certain types of devices such as wireless phones increases.

A mini-cookie is a small cookie, approximately 10 bytes in size that contains a token which can be used to reference session information in SiteMinder in-memory session store. The mini-cookie is a fraction of the size of a standard SiteMinder cookie, and provides an alternative for standard SiteMinder cookies.
Mini-cookie Session Scheme Configuration

The mini-cookies session scheme stores session information in the SPS in-memory session store and creates a cookie that contains an encrypted token that the SPS returns to the user.

This section has the following format:

```xml
<SessionScheme name="minicookie">
  class="com.netegrity.proxy.session.MiniCookieSessionScheme"
  accepts_smsession_cookies="false"
  # The name of the small cookie to be stored in the client.
  cookie_name="SMID"
</SessionScheme>
```

The directives in the mini-cookies session scheme are listed following.

**class**

Specifies the java class that defines the session scheme. This directive is not modified when you want to use the mini-cookies session scheme provided with the SPS.

**Default:** com.netegrity.proxy.session.MiniCookieSessionScheme

**accepts_smsession_cookies**

Indicates if SMSESSION cookies are accepted. Specify one of the following values:

- **true**
  
  Indicates that SMSESSION cookies are accepted and used by the session scheme.

- **false**
  
  Indicates that the SMSESSION cookies are not supported by the session scheme. Use this setting to verify that only a mini-cookie session is used for the session scheme.

**cookie_name**

Indicates the name of the mini-cookie that contains the token for the user session.

**Note:** This name is not configured using the same value for all SPS that provides single sign-on.

Simple URL Rewriting Session Scheme

Simple URL rewriting is a method for tracking a user session by appending a token to the requested URL. This token is used to retrieve session information from the in-memory session store.
Simple URL Rewriting Configuration

The simple_url schemes support simple URL rewriting, which can be accomplished without any custom work.

**Note:** The CGI-based and FCC-based password schemes are supported with the simple_url session scheme.

**Example**

A user accesses a host and the user session is established through the simple URL rewriting session scheme. An initial request can look like the following example:

http://banking.company.com/index.html

If the user provides appropriate credentials and is authenticated and authorized, the URL requested by the user is rewritten and returned to the user in a form similar to the following:

http://banking.company.com/SMID=nnnnnnnnnn/index.html

$nnnnnnnnnn$  
Represents a hashed, randomly generated token that the SPS uses to identify the user session.

**Important!** For the simple URL rewriting session scheme to work, any links defined in the enterprise must be relative links. If links are absolute, the simple URL rewriting scheme fails. Also, the token that the SPS appends to a URL is stripped from the URL when the request is forward. The token is only appended at the SPS interaction level so that it does not interfere with back-end server processing.

The format of the SimpleURL scheme is:

```xml
<SessionScheme name="simple_url">
    class="com.netegrity.proxy.session.SimpleURLSessionScheme"
    accepts_smsession_cookies="false"
    session_key_name="SMID"
</SessionScheme>
```

The directives in the SimpleURL scheme are listed following.

**class**

Specifies the Java class that defines the session scheme. This directive is not modified when you want to use the cookieless rewriting session scheme.

**Default:** com.netegrity.proxy.session.SimpleURLSessionScheme
accepts_smsession_cookies

Indicates whether SMSESSION cookies are accepted. Specify one of the following values:

true

Indicates that SMSESSION cookies are accepted and used by the session scheme.

false

Indicates that SMSESSION cookies are not supported by the session scheme. Use this setting to verify that only a cookieless rewriting session is used for this session scheme.

session_key_name

Specifies the SiteMinder ID (SMID) session identifier.

Note: When a cookieless federation transaction is being processed by the SPS federation gateway and the simple_url session scheme is used, the SMID is added to the request as a query parameter instead of the being appended to the URI.

Enable Cookieless Federation for Rewriteable Session Schemes

For the SPS to use rewritable session schemes, such as simple URL session scheme, in a federated environment, configure cookieless federation.

To configure cookieless federation

1. Open the server.conf file in a text editor. This file is located in the directory
   sps_home/secure-proxy/proxy-engine/conf
2. Add the following code to the virtual host section for the virtual host that is serving FWS.
   
   cookielessfederation="yes"
3. Save the file.
4. Restart the SPS.

Note: No separate post filter, such as the CookielessFedFilter needs to be enabled for the SPS federation gateway. This functionality is provided out-of-the-box when you enable the federation gateway functionality. You have to enable this post filter when the SPS is not acting as a federation gateway.
Rewrite FWS Redirects for Simple URL Session Schemes

If you deploy the SPS in a federated environment, one of the session schemes you can at the site producing assertions is a simple URL session scheme. If you use this scheme, you may be required to rewrite the links that direct the user to the appropriate site so that the session key is added to the link. In SiteMinder documentation, these links for SAML 1.x are called intersite transfer URLs. For SAML 2.0, these links are referred to as an unsolicited response or an AuthnRequest link.

For rewriting the links so that the session key information is added to the base of the URLs, a sample post filter, RewriteLinksPostFilter, is provided along with the SPS filter examples. This filter can be compiled and be attached to the appropriate proxy rule, which handles the forwards to the intersite transfer URL, unsolicited response, or AuthnRequest.

The RewriteLinksPostFilter provided with the SPS is a sample filter. You must configure the filter to suit your requirements.

Note: If you use the simple_url session scheme for transactions involving the SPS federation gateway, the session key (SMID) gets added to the request as a query parameter instead of being appended to the URI. However, the SMID gets added to the URI when the final target resource is accessed at the back-end server.

Wireless Device ID Session Scheme

Some wireless devices have a unique device identification number. This number is sent as a header variable with any requests for resources. The SPS can use this device ID as a token to refer to session information in the session store.

Because device IDs differ by wireless device vendor, define a device ID session scheme in the server.conf file. This scheme must include the class information and a device_id_header_name directive that is set to the vendor-specific device ID.

The format of the device ID scheme is:

```
<SessionScheme name="device_id">
    class="com.netegrity.proxy.session.DeviceIdSessionScheme"
    accepts_smsession_cookies="false"
    device_id_header_name="vendor_device_id_header_name"
</SessionScheme>
```
Uses for Each Session Scheme

The following table illustrates the scenarios in which each session scheme is used. The session schemes are based on the sensitivity of resources on a virtual host.

<table>
<thead>
<tr>
<th>Session Scheme</th>
<th>Security Level</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL Session ID</td>
<td>High</td>
<td>This scheme provides a clean and highly secure means of holding user sessions. Although the most secure of the available schemes, it is limited in scalability. All content must be served over SSL and the user must continue to access the same SPS server for the session to persist. Also, some browsers (some versions of IE), by default, terminate the SSL session after 2 minutes. This scheme is ideal for intranet and extranet applications with high security needs.</td>
</tr>
<tr>
<td>SiteMinder Cookies</td>
<td>Medium or High</td>
<td>This scheme is the traditional SiteMinder session mechanism, which has proven highly secure in many enterprise deployments. For maximum security, the WebAgent SecureCookie setting are set to &quot;Yes&quot;.</td>
</tr>
<tr>
<td>IP Address</td>
<td>Low</td>
<td>This scheme is only for applications where users are retrieving information (with HTTP GET) from protected resources and not sending (with HTTP POST) information to a secure application. An example of an appropriate application would be an online library. An example of an in-appropriate application would be a bond trading application.</td>
</tr>
<tr>
<td>Mini-Cookies</td>
<td>Medium or High</td>
<td>This scheme is ideal for applications where user clients accept cookies but are accessing the application over connections of limited speed and bandwidth. For maximum security, the WebAgent SecureCookie setting is set to &quot;Yes&quot;.</td>
</tr>
<tr>
<td>Simple URL Rewriting</td>
<td>Medium</td>
<td>This scheme is ideal for environments that do not support or want to use cookies.</td>
</tr>
<tr>
<td>Device ID</td>
<td>Medium</td>
<td>This scheme is designed for wireless environments where a device ID is sent with every client request to identify a user.</td>
</tr>
</tbody>
</table>
Multiple Session Schemes for Virtual Hosts

The SPS supports multiple session schemes for each virtual host in an enterprise. Session schemes can be assigned to each user agent type that has access to a virtual host. The following illustration shows the SPS configured for four virtual hosts.

The preceding illustration shows that the session scheme varies for a user agent depending on the virtual host accessed by the user. For example, when accessing www.company.com with a browser, the SPS uses SiteMinder cookies to maintain a user session. But when accessing BondTrading.company.com, browser sessions are maintained using the SSL ID of the user HTTPS connection. The sessions for PDA and wireless users are maintained using cookieless session schemes, while browser user sessions are maintained through cookies or mini-cookies.
Deleting Attribute Cookies for Cookieless Federation

To support environments that do not want to exchange cookies, the SPS maps the SiteMinder session cookies to a cookieless session scheme and deletes the cookies from the response. However, the attribute cookies are not mapped and remain in the response.

The FWS application processes the federation requests and inserts attribute cookies and SiteMinder-created session cookies in its responses. The responses are forwarded to the SPS.

The SPS can be configured to delete the attribute cookies inserted by the FWS application.

**To configure the SPS to delete session and attribute cookies**

1. Open server.conf file in a text editor.
   
   The file is located in the directory `sps_home/secure-proxy/proxy-engine/conf`.

2. Add the following code to the virtual host section for the virtual host that is serving the FWS.

   ```
   deleteallcookiesforfed="yes"
   ```

3. Save the file.

4. Restart the SPS.

User Agent Settings in the Server.conf

User agents define the device types such as web browsers, wireless phones, and PDAs by which users can access network resources. All user agents must be defined in `<UserAgent>` elements. Each `<UserAgent>` element includes a name attribute that identifies the user agent type. By default no user agent types are predefined in the server.conf file.

The user agent configuration section has the following format:

```
#<UserAgent name="user_agent_name_1">
#  header_name_1=some regular expression
# </UserAgent>
```
The directive in the UserAgent section is:

**header_name**

This directive contains the user-agent header of an HTTP request. This header indicates the type of device making the request. You can use regular expressions and provide a partial name as part of the expression. This allows you to specify user agent types whose user-agent header may contain slight differences, such as version numbers.

Device types must be defined in a `<UserAgent>` element before they can be associated with a session scheme in a `<SessionSchemeMapping>` element.

**Nokia User Agent Settings**

You can specify a Nokia user agent type, which is for Nokia wireless phones. The name attribute for the `<UserAgent>` section is the name used to identify the user agent type when specifying session scheme mappings.

The Nokia user agent entry format would be as follows:

```
# Nokia
<UserAgent name="Nokia">
  User-Agent="Nokia."
  attribute_name="value"
</UserAgent>
```

The directives for the Nokia user agent are as follows:

**User-Agent**

This directive contains the contents of the user agent header of an HTTP request. This header indicates the type of device making the request. You can use regular expressions and provide a partial name as part of the expression. This directive allows you to specify user agent types whose User-Agent header can possibly contain slight differences, such as version numbers.

Default: Nokia

**attribute_name**

The sections of the server.conf for wireless devices and other user agent types can possibly contain additional attributes and values for those attributes. Attributes are not required, but can possibly be desirable for some applications.
Virtual Host Settings in the server.conf File

The `<VirtualHostDefaults>` element of the server.conf file specifies default settings for virtual hosts. These settings are used for each virtual host you add to the SPS.

To specify non-default values for a virtual host, add a `<VirtualHost>` element following the `<VirtualHostDefaults>` element. The `<VirtualHost>` element must contain directives and values that are different from the default virtual host.

The default virtual host settings are broken up into the following sections:

- Default session scheme
- Session scheme mappings
- WebAgent.conf settings
- Default virtual host name

The format of the `<VirtualHostDefaults>` section is as follows:

```xml
<VirtualHostDefaults>
    # default session scheme
    defaultsessionscheme="default"
    enablerewritecookiepath="no"
    enablerewritecookiedomain="no"
    enableproxypreservehost="no"

    # specify the block size for request and response in KBs
    requestblocksize="4"
    responseblocksize="4"

    #TO-DO: Define any session scheme mappings
    <SessionSchemeMappings>
        # user_agent_name=session_scheme_name
    </SessionSchemeMappings>
</VirtualHostDefaults>
```

# Web Agent.conf

```xml
<WebAgent>
sminitfile="C:\Program Files\netegrity\secure-proxy\proxy-engine\conf\defaultagent\WebAgent.conf"
</WebAgent>
</VirtualHostDefaults>
```
Setting Virtual Host Cookie Path and Domain to the Correct URI

The virtual host configuration in the server.conf file includes the enablerewritecookiepath and enablerewritecookiedomain parameters, which you can use to manage cookies generated by a destination server that sits behind the SPS. When the SPS receives a request from a client, the SPS authenticates the user and directs the client to the requested destination server. The destination server generates a cookie that it places in the browser, then the SPS sends the user back the client response with the cookie. After receiving the response from the SPS, the client stores the cookie.

When the client sends a subsequent request, the browser retrieves the stored cookie associated with the URL. In some cases, the destination server can possibly have set the cookie path to its own resource URI and not to the URI of the initial request. As a result, when the client sends the subsequent request, the browser contains the wrong cookie or does not even have a cookie. The request is received at the destination server with the wrong cookie or with no cookie at all.

To help ensure that the correct cookie is set in the browser, you can configure the SPS to rewrite the cookie path and cookie domain. The destination server sets the cookie path and cookie domain to the URI of the resource on the SPS server. The client can send the correct cookie back with subsequent requests to SPS.

The two parameters operate as follows:

**enablerewritecookiepath**

Instructs the SPS to rewrite the cookie path to the URI of the initial request from the URI set by the server that sits behind the proxy.

**Default:** no

**enablerewritecookiedomain**

Instructs the SPS to rewrite the cookie domain from the domain set by the server sitting behind the proxy to the domain of the initial request.

**Default:** no

**Example**

The client requests an SPS resource http://mysps.ca.com/basic/test/page0.html. With the enablerewritecookiepath set to yes, the cookie path is rewritten to /basic/test before the browser is sent back to the client. This cookie is rewritten regardless of the cookie path that was originally in the cookie received by SPS from the destination server.
Virtual Host Settings in the server.conf File

To rewrite backend cookie paths and domain
1. Open the server.conf file in a text editor.
2. Set one or both of the following parameters to yes:
   - enablerewritecookiepath
   - enablerewritecookiedomain
3. Save the file.
4. Restart the SPS.

Preserve the HOST Header File

You can preserve the HTTP HOST header file and instruct the SPS to send it to the backend by setting the value of the enableproxypreservehost parameter to YES. This parameter is located in the <VirtualHostDefaults> section of the server.conf file. Its default value is NO.

Handling Large Files Using Data Blocks

The SPS handles the transfer of large files to users by breaking up the data transferred between the SPS and the backend server into blocks. You control the block size read by the SPS using two parameters in the Virtual Host section of the server.conf file:

- requestblocksize
- responseblocksize

When a user sends a file to a backend server, the SPS verifies the requestblocksize specified for that virtual host. Based on the value of requestblocksize, the SPS breaks down the data into blocks and then forwards the blocks to the backend server.

Similarly, when the backend server sends data to the user, the SPS verifies the responseblocksize specified for that virtual host. Based on the value of responseblocksize, the SPS reads the data in blocks from the backend server before further processing the blocks. This enables the user to control the number of read-write operations for such file transfers. To handle large file transfers, use large block sizes.

Note: The requestblocksize and responseblocksize parameters should be defined in proportion to the available and allocated JVM heap size for the SPS java process.
Define File Data Block Size for Large File Handling

To define the block size to handle large files when you configure the block size for a virtual host, modify the request and response block sizes for each virtual host. These parameters are valid only for that virtual host. The data block sizes can be different for different virtual hosts, but the settings only apply to the associated virtual host you are configuring.

To define file size for large file handling

1. Open the server.conf file in a text editor.
2. Edit the following parameters under the virtual host configuration:
   
   **requestblocksize**
   
   Defines the block size of the request data that will be read at a time before the data blocks are sent to the backend server. The block sizes are in KB.

   **Limits:** 1KB to approximately 352000 KB. If the value is set to 0 KB, the transaction hangs without an error posted to the log files. For any value greater than or equal to 8 KB, chunks of 8 KB are created. A corresponding chunk size is create for values between 1 KB and 8 KB.

   **Default:** 4

   **responseblocksize**

   Defines the block size of the response data that will be read at a time before the data blocks are forwarded from the backend server to the user. The block sizes are in KB.

   **Limits:** 1KB to approximately 352000 KB. If the value is set to 0 KB, the transaction hangs without an error posted to the log files.

   **Default:** 4

3. Save the server.conf file.
4. Restart the SPS.
Adjust JVM Heap Size for Data Blocks

The requestblocksize and responseblocksize parameters are defined in proportion to the available and allocated JVM heap size for the SPS java process.

To define SPS JVM heap size

1. Navigate to the appropriate directory:
   - Windows: sps_home/secure-proxy/proxy-engine/conf
   - UNIX: sps_home/secure-proxy/proxy-engine

2. Open one of the following files
   - For Windows systems: SmSpsProxyEngine.properties file
   - For UNIX systems: proxyserver.sh file

3. Add the following parameters in the Java section of the file:
   - -Xms256m
   - -Xmx512m

4. Save the file.

Session Scheme Mapping for the Default Virtual Host

Session scheme mappings associate session schemes with user agent types. The user agent types defined in <UserAgent> elements of the server.conf file must be mapped to session schemes defined in the <SessionScheme> elements.

The format of the session scheme mapping associated with user agents is as follows:

```xml
#<SessionSchemeMappings>
    #    user_agent_name=session_scheme_name
#</SessionSchemeMappings>
```
The directive in this section is:

```plaintext
user_agent_name
```

Associates a user agent with a session scheme. To set the values:

```plaintext
user_agent_name
```

Specifies a name defined in a `<UserAgent>` section of the file.

```plaintext
session_scheme_name
```

Specifies a scheme defined in a `SessionScheme` element.

**Example:**

User Agents named browser, phone1, and phone2 have been defined and mapped to any of the session schemes defined in the file. For this example, browser is mapped to the default session scheme, phone1 is mapped to the simple_url scheme, and phone2 is mapped to the minicookie session scheme.

The resulting `<SessionSchemeMappings>` element appears as follows:

```plaintext
# Session Scheme Maps
<SessionSchemeMappings>
  browser="default"
  phone1="simple_url"
  phone2="minicookie"
</SessionSchemeMappings>
```

---

**Web Agent Settings for the Default Virtual Host**

The `server.conf` file includes a `<WebAgent>` section for the `<VirtualHostDefaults>`. The `sminitfile` directive specifies the configuration file, `WebAgent.conf` for the default web agent. If local configuration is allowed, the `WebAgent.conf` file points to a local configuration file, `LocalConfig.config`.

If you create more than one virtual host, you can use the default Web Agent when you do not intend to use alternate settings in the Web Agent configuration file. If you plan to set any directive differently, for example, to specify a different log level, use a different Web Agent for the new virtual host.

**To configure a new Web Agent for a new virtual host**

1. Create a directory with the name of the new virtual host, for example, `serverb`.

2. Copy the contents of the directory for the default virtual host into the new directory.

Run `smreghost` if the new Web Agent points to a different SiteMinder installation. **Note:** If the Web Agent configuration objects for both virtual hosts point to the same SiteMinder installation, you do not need to run `smreghost`. You can use the same `smhost` file for both the Web Agents.
3. Use a text editor to modify WebAgent.conf to reflect the new agent configuration object. Verify that the Web Agents have different log files.

4. Open the WebAgent.conf file and add the following required directive with a unique value.

   ServerPath="path"

   path
   Specifies is the fully qualified path to the WebAgent.conf file you are editing
   ■ For Windows, this value must be a unique alphanumeric string. The backslash ‘\’ character is not permitted in this string.
   ■ For UNIX, this value must be the fully qualified path to the WebAgent.conf file you are editing.

5. Access the Agent Configuration Object at the Policy Server that corresponds to the first host configuration object in the server.conf file. Verify the Agent cache settings for MaxResourceCacheSize and MaxSessionCacheSize and also that the cache limits take into account all Agent Configuration Objects.

   Note: For detailed information about the Web Agent settings, see the CA SiteMinder Web Agent Guide.

Setting the requirecookies Parameter

The requirecookies setting in the server.conf file is a special Web Agent 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 to process HTTP requests successfully, including basic Authorization headers.

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

Set the requirecookies setting to yes when all user agent types for the associated virtual server use the default session scheme. If an agent type uses a cookieless session scheme, set the requirecookies parameter to no.
Handling Redirects by Destination Servers

Some destination servers can respond to a request from the SPS with a redirection.

**Note:** A redirection that is the result of a request to the SPS is not the same as a redirect that occurs in a proxy rule. For information about a redirect in a proxy rule, see `nete:redirect`.

Because the redirection initiated by the destination server is likely to a server behind the DMZ, the URL specified in the redirection results in an error. However, you can include parameters in a virtual host configuration that substitute the virtual host server name and port number in place of a redirect from a destination server.

To substitute the virtual host server and port for redirect writing, configure the following:

**enableredirectrewrite**

Enables or disables redirect rewriting. If this directive is set to a value of `yes`, the URL for a redirect initiated by a destination server is examined by the SPS. If the redirect URL contains a string found in the list of strings specified in the associated `redirectrewritablehostnames` parameter, the server name and port number of the redirect are replaced by the server name and port number of the virtual host.

If the parameter is set to a value of `no`, any redirects initiated by destination servers are passed back to the requesting user.

**redirectrewritablehostnames**

Contains a comma-separated list of strings that the SPS searches for when a redirection is initiated by a destination server. If any of the specified strings are found in the server or port portion of the redirect URL, the SPS substitutes the name and port number of the virtual host for the server name and port portion of the redirect URL.

If you specify a value of "ALL" for this parameter, the SPS substitutes the server name and port number of the virtual host for all redirects initiated by the destination server.
For example, consider a virtual host configuration in the server.conf file that contains the following parameters:

```xml
<VirtualHost name="sales">
  hostnames="sales, sales.company.com"
  enableredirectrewrite="yes"
  redirectrewritablehostnames="server1.company.com,domain1.com"
</VirtualHost>
```

When a user makes a request from http://sales.company.com:80, the SPS forwards the request to a destination server according to proxy rules. If the destination server responds with a redirect to server1.internal.company.com, the redirect is rewritten before being passed to the user as sales.company.com:80.

**Note:** The proxy rules for the SPS must be configured to handle the redirected requests.

**Virtual Host Names Configuration**

For the SPS to act as a virtual host for one or more host names, configure a `<VirtualHost>` element for related hostnames and IP addresses. Each server.conf file must contain one `<VirtualHost>` element for the default virtual host, in addition to additional elements for hostnames at other IP addresses.

The following is an example of a `<VirtualHost>` element for the default virtual host in a server.conf file.

```
# Default Virtual Host
<VirtualHost name="default">
  hostnames="home, home.company.com"
  addresses="123.123.12.12"
</VirtualHost>
```

The default virtual host in the preceding example is the host named home.company.com residing at IP address 123.123.12.12. You can add hostnames that resolve as the default virtual host by adding to the comma-separated list of values in the hostnames. To add additional virtual hosts to the proxy configuration, add additional `<VirtualHost>` elements that include hostname directives for the additional virtual hosts.

**Example:**

To add a sales virtual host for the server, sales.company.com virtual host, add the following element:

```
<VirtualHost name="sales">
  hostnames="sales, sales.company.com"
</VirtualHost>
```
Override Default Values for Virtual Hosts

The <VirtualHostDefaults> settings are used for all virtual hosts defined in the server.conf file unless you explicitly enter settings for a particular virtual host.

You do not have to reconfigure all the virtual settings for the single virtual host. Any settings that you do not redefined in the <VirtualHost> element are applied from the <VirtualHostDefaults> settings.

To override virtual host default values

1. Add any directive in the default virtual host configuration to the <VirtualHost> element that you want to modify.
2. Specify a new value for the directive in the <VirtualHost> element.
3. Save the file.
4. Restart the SPS.

Example

The virtual host named "sales" requires a default session scheme from what is configured for the default virtual host. The <VirtualHost> element could be modified as follows:

```
<VirtualHost name="sales">
  hostnames="sales, sales.company.com"
  addresses="123.123.22.22"
  defaultsessionscheme="minicookie"
</VirtualHost>
```
Chapter 9: Configuring Proxy Rules

This section contains the following topics:

- **Proxy Rules Overview** (see page 117)
- **Establish a Proxy Rules Configuration File** (see page 120)
- **Proxy Rules DTD** (see page 120)
- **How nete:xprcond Elements Works** (see page 132)
- **Regular Expression Syntax** (see page 132)
- **Header Values in Forwards, Redirects, and Results Filters** (see page 136)
- **Response Handling** (see page 138)
- **Modify Proxy Rules** (see page 138)
- **Sample Proxy Rules Configuration Files** (see page 138)

### Proxy Rules Overview

The primary purpose of the SPS is to route requests to the appropriate destination servers in the enterprise. The SPS routes requests using the proxy engine, which is built into the server. The proxy engine interprets proxy rules and provides both a forward and a redirect service to handle the disposition of all user requests for back end resources.

Proxy rules define the details of how the SPS forwards or redirects requests to resources located on destination servers within the enterprise. A set of proxy rules is defined in an XML configuration file according to the proxy rules DTD, which is installed with the SPS.

**Note:** The proxyrules.xml file contains a default rule that forwards requests to http://www.ca.com$0, where $0 appends the entire URI from the original request to the destination, which is www.ca.com in this case. You must modify this rule to suit your environment.

More information:

- **Establish a Proxy Rules Configuration File** (see page 120)
- **Proxy Rules DTD** (see page 120)

### Planning Routes for Incoming Requests

Before you set up the proxyrules.xml file, you should map out the routing for incoming requests. Depending on the virtual host that contains the requested resource, the proxy rules can use a default destination; forward a request based on the user agent type, or uses a HTTP header value to determine the destination server that will fulfill the request. The SPS can provide access to a number of virtual hosts.
The following illustration shows how proxy rules can be used to route requests to appropriate destination servers.

Note: A proxy rules configuration file is processed from top to bottom by the SPS. The first condition that is met by an incoming request determines the behavior of the proxy engine. For example, if a set of proxy rules has two conditions based on a string contained in the URI of a request, and part of the URI of an incoming request matches both of the strings, the first condition listed in the proxy rules file will be used to route the request.

Proxy rules can also be used to control more complex conditions for directing requests to destination servers.

The following illustration shows how proxy rules can be used with a second level of conditions nested within parent conditions.
Proxy Rules Terminology

The proxy rules configuration file is a description of the XML elements that the SPS uses when routing user requests. The following terms are used to describe the proxy rules:

Destinations

A destination is a URL that fulfills a request. The SPS forwards a request to a destination, or sends a redirect response to a user that specifies a destination. A set of proxy rules must contain destinations that can be reached according to the conditions and cases defined in the proxy rules.

Conditions

A condition is an attribute of a request that allows the SPS to determine the destination of a request. Each condition may have many cases that the SPS evaluates to direct a request to the appropriate destination. Each condition must contain a default element that defines the behavior if a request does not match any of the cases defined in the condition.

The conditions may include any one or more of the following:

URI

The SPS uses the portion of the requested URL after the hostname to determine how to route a request. Using criteria described in the DTD, a portion of a URI, such as the file extension of the requested resource, can be used to route requests.

Query String

The SPS uses the query string portion of a URI to determine how to route a request. The query string includes all the characters following a '?' in the request.

Host

The SPS uses the requested server hostname to determine how to route a request. The port number of the hostname can also be used as criteria for routing requests. This condition is used when the proxy has more than one virtual server.

Header

Uses the value of any HTTP header to determine how to route a request. To route requests based on the type of device being used to access resources, requests may be routed according to the USER_AGENT HTTP header.

Note: HTTP headers derived from SiteMinder responses may be used to determine how to route a request.
Establish a Proxy Rules Configuration File

Cases

A case is a set of specific values for conditions that provide the information the SPS needs to determine the ultimate destination for a request. For example, if a set of proxy rules uses the host condition, cases include the virtual hosts configured for the system, such as home.company.com and banking.company.com.

Establish a Proxy Rules Configuration File

The proxy rules configuration file is an XML configuration that is identified by the rules_file directive of the <ServiceDispatcher> element in the server.conf file. The rules_file directive indicates the relative path from the installation directory to the proxy rules configuration file. At installation, the relative path to the default proxy rules configuration file is generated automatically and inserted into the rules_file directive for the default virtual host.

The generated path and proxy rules file name is the following:

```
sps_home/secure-proxy/proxy-engine/conf/proxyrules.xml
```

Changes to the proxy rules configuration file do not require a server restart to take affect; the SPS detects when changes are made to the file and loads the new proxy rules file.

If the SPS detects an error in the proxy rules when parsing the rules, the SPS records an error in the server log, ignores the changes, and uses the existing proxy rules. The server log file location is specified in the server.conf file.

More information:

General Server Settings in the server.conf File (see page 74)
Service Dispatcher Settings in the server.conf File (see page 86)

Proxy Rules DTD

You must create the proxyrules.xml file using the proxy rules DTD. To view the proxy rules DTD, go to the following directory:

```
sps_home\secure-proxy\proxy-engine\conf\dtd
```

The following elements are configurable in the DTD:

- nete:proxyrules
- nete:description
- nete:case
The full definition for the nete:proxyrules element is:

```xml
```

This element is the root element for a proxy rules XML configuration file. It contains an optional nete:description element and one of the following elements:

- nete:cond
- nete:xprcond
- nete:forward
- nete:redirect
- nete:local

The nete:proxyrules element must be present in a proxy rule configuration file.

### Debug Attribute

The nete:proxyrules element has the following attribute:

```xml
<ATTLIST nete:proxyrules debug (yes|no) "no"
```

This attribute enables or disables logging that aids in debugging proxy rules. The default for this attribute is no. To enable logging, set this attribute to yes.

For example:

```xml
<nete:proxyrules xmlns:nete="http://www.company.com/" debug="yes">
```

When enabled, trace logging information for the SPS is included in trace logs. The location of the log file is determined by the TraceFileName parameter in the agent configuration object that you specified during the SPS installation process. The TraceConfigFile parameter in the same agent configuration object must point to the Secure Proxy-specific trace logging configuration file.
By default this file is located in:
<install-dir>\proxy-engine\conf\defaultagent\SecureProxyTrace.conf

**Note:** This change does not require a SPS restart for the changes to take effect.

**nete:description**

The full definition for the nete:description element is:

```xml
<!ELEMENT nete:description (#PCDATA)>
```

This is an optional element that contains a parsed character data (PCDATA) description of another element.

**Note:** PCDATA is any text that is not markup text.

A nete:description element can be a child of the nete:proxyrules element and may contain a description of your choosing.

**nete:case**

The nete:case element provides the destination associated with a specific value for a condition defined in the nete:cond parent element. The SPS uses the value of the nete:case element to evaluate a case.

The definition for the nete:case element is:

```xml
```

All nete:case elements must contain one of the following child elements:

**nete:cond**

nete:case elements can contain additional nete:cond elements. This allows you to nest multiple conditions in a set of proxy rules.

**nete:xprcond**

nete:case elements can contain additional nete:xprcond elements for regular expression matching of URIs. This allows you to nest a regular expression condition in a set of other conditions.
nete:forward

Provides a destination to which requests that fulfill the nete:case comparison will be forwarded.

nete:redirect

Provides a destination to which requests that fulfill the nete:case comparison will be redirected. Once redirected, requests are fulfilled directly by the destination server, rather than through the SPS.

In the following example, a nete:cond element specifies URI matching, and the nete:case elements demonstrate possible uses of the comparison type attribute.

```xml
<nete:cond type="uri" criteria="beginswith">
  <nete:case value="/hr">
    <nete:forward>http://hr.company.com$0</nete:forward>
  </nete:case>
  <nete:case value="/employee">
    <nete:forward>http://employees.company.com$1</nete:forward>
  </nete:case>
</nete:cond>

Note: The <nete:forward> URL </nete:forward> elements must be located on the same line. In the example, the </nete:forward> closing tags sometimes appear on a separate line due to space constraints, however, a line break in an actual proxy rules file causes an error. The SPS interprets line breaks before the </nete:forward> closing tag as characters that are part of the URL contained in the nete:forward element.
Forward and Redirect Syntax

When forwarding or redirecting a request, the SPS uses a system for maintaining some part or all of the Universal Resource Indicator (URI) specified by a user. This URI points to a resource that lies on a destination server and must be interpreted by the SPS to fulfill a request.

Either of the following may be appended to a URL specified in a forward or redirect destination:

\$0

Appends the entire URI string from a user’s request to the destination specified in the proxy rule.

For example, if a proxy rule forwards all user requests for www.company.com to proxy.company.com$0 and a user request for proxy.company.com/employees/hr/index.html, that request is forwarded to www.company.com/employees/hr/index.html.

\$1

May only be used in nete:case elements where the parent nete:cond element specifies a URI substring match using the begins with comparison. $1 indicates that everything to the right of the matching text is appended to the forwarded or redirected request.

For example, consider a proxy rules configuration file that has a nete:cond element of:

```
<nete:cond type="uri" criteria="beginswith">
  <nete:case value="/hr">
    <nete:forward>http://hr.company.com$1</nete:forward>
  </nete:case>
</nete:cond>
```

Assume this condition is the child of a condition that is evaluating URIs for a hostname of www.company.com and a nete:case element of:

```
<nete:case value="/hr">
    <nete:forward>http://hr.company.com$1</nete:forward>
</nete:case>
```

If a user requests:

http://www.company.com/hr/employees/index.html

The request is forwarded to:

http://hr.company.com/employees/index.html

**Note:** Because this example specifies the $1 parameter, the /hr portion of the URI is omitted when the request is forwarded to hr.company.com.
The definition for the nete:cond element is:

```
<!--ELEMENT nete:cond (nete:case+,nete:default?)-->
```

In addition, the nete:cond element has the following attributes:

```
<!--ATTLIST nete:cond type (header | host | uri | query) #REQUIRED
 criteria (equals | beginswith | endswith | contains) "equals"
 headername CDATA #IMPLIED-->
```

The `nete:cond` element specifies a condition that will be evaluated to determine how the SPS handles incoming requests. This element must include an attribute to be evaluated by the SPS.

Possible attributes, as defined in the `ATTLIST` element, include:

**header**

Specifies an HTTP header. HTTP headers are name-value pairs which can be retrieved from a user directory by SiteMinder. If you select header as the type, you must also specify the name of the header. The following is an example of a `nete:cond` element using header as the type:

```
<nete:cond type="header" headername="USER_AGENT">
```

This element indicates that a header will be used to determine the destination of a request, and that the header to be evaluated by the SPS is `USER_AGENT`. The actual destinations for requests are determined by `nete:case` elements which are children of `nete:cond` elements as described in the next section.

**Note:** HTTP headers generated by SiteMinder responses may be specified in `nete:cond` elements.

**host**

Specifies a host name in deployments where the SPS proxies for multiple virtual hosts.

Port numbers are considered part of the host, so you can use the `endswith` criteria, described later, in conjunction with the host condition to route requests by port number.
query

Specifies the query string portion of the URI that follows the '?' character. This is similar to a nete:cond that makes use of the URI as follows:

URI

Specifies universal resource indicator, which is the portion of a requested URL that follows the server name.

You can use the endswith criteria in conjunction with the URI condition to route requests by file extension.

One of the type attributes described above must be included in the nete:cond element. In addition, the nete:cond element must specify a criteria that defines the comparison that the proxy engine executes on the value of the condition for an incoming request.

Possible criteria are:

equals

Indicates that the value of the type attribute of the nete:cond parent element must equal the contents of the value attribute of the nete:case element to act on a request.

beginswith

Indicates that the value of the type attribute of the nete:cond parent element must begin with the contents of the value attribute of the nete:case element to act on a request.

endswith

Indicates that the value of the type attribute of the nete:cond parent element must end with the contents of the value attribute of the nete:case element to act on a request.

contains

Indicates that the value of the type attribute of the nete:cond parent element must contain the contents of the value attribute of the nete:case element to act on a request.

Note: If no criteria are specified, the SPS assumes the default criteria of equals.
All nete:cond elements except those using the header type take the following form:

```xml
<nete:cond type="condition_type" criteria="equals|beginswith|endswith|contains">

condition_type
  Indicates a condition type of host, URI, query, or header.

For example: <nete:cond type="host" criteria="equals">

Note: Conditions that use headers as the comparison require an additional argument of headernamewithout="value" where value is the name of the HTTP or SiteMinder header.

Finally, each nete:cond element must have one or more nete:case child elements. The nete:case children provide the unique values that the SPS uses to route requests to appropriate destinations.

**nete:default**

The definition of the nete:default element is:

```xml
```

This element is required and must be a child element of each nete:cond element. If a request does not meet the requirements of any nete:case elements contained in nete:cond elements, the nete:default element determines how to handle the request.

The possible child elements associated with the nete:default element are identical to the elements available for a nete:case element. If you create nete:cond elements as children to a nete:default, be careful to take into account a default case so that all possible client requests may be handled by the SPS.

In the following example, the nete:default element forwards all requests that do not meet the criteria of any other proxy rules to a home page of general information.

```xml
<nete:default>
  </nete:forward>
</nete:default>
```

The opening and closing tags, `<nete:forward>URL</nete:forward>`, elements must be located on the same line. In the example, the `</nete:forward>` closing tags sometimes appear on a separate line due to space constraints, however, a line break in an actual proxy rules file causes an error. The SPS interprets line breaks before the `<nete:forward>` closing tag as characters that are part of the URL contained in the nete:forward element.
The definition of the nete:forward element is:

```xml
<!ELEMENT nete:forward (#PCDATA)>  
```

The nete:forward element forwards a request to a specified URL.

**Note:** The `<nete:forward>` and `</nete:forward>` tags must be located on a single line in the proxy rules file. If they are not located on the same line, the SPS interprets line breaks as part of the URL contained in the element. This causes the forward service to fail.

In the following example, the nete:forward element forwards requests while maintaining the URI requested by the user.

```xml
<nete:forward>http://home.company.com$0</nete:forward>
```

If the user’s request satisfies the nete:case parent element’s criteria, that request is forwarded to home.company.com. Therefore, a request for `http://server.company.com/hr/benefits/index.html` that is forwarded by the previous nete:forward element is fulfilled by forwarding the request to `http://home.company.com/hr/benefits/index.html`.

If you want to forward a request over SSL, be sure to use https instead of http when defining the destination contained in the `<nete:forward>` element.

The nete:forward element contains the following attribute:

```xml
<!ATTLIST nete:forward filter CDATA #IMPLIED>
```

This attribute allows you to specify the name of a Java filter class that can be invoked during a forward from the SPS to a destination server. Filters can be written using the Filter API.

**More information:**

- Forward and Redirect Syntax (see page 124)
- Filter API Overview (see page 182)

The nete:forward element contains the following attribute:

```xml
<!ATTLIST nete:forward filter CDATA #IMPLIED>
```

This attribute allows you to specify the name of a java filter class that can be invoked during a forward from the SPS to a destination server. Filters can be written according to the Filter API.
**nete:redirect**

The definition of the nete:redirect element is:

```xml
<!ELEMENT nete:redirect (#PCDATA)>
```

The nete:redirect element specifies a response that is returned to a user which redirects the user request to an appropriate destination server. The PCDATA follows the standard forward and redirect syntax. Once redirected, the SPS does not handle the completion of the request. Instead, the request is handled by the server that is the target of the redirection.

The `<nete:redirect>` and `</nete:redirect>` tags must be located on a single line in the proxy rules file. If they are not located on the same line, the SPS interprets line breaks as part of the URL contained in the element. This causes the redirect service to fail.

In the following example, the nete:redirect element redirects requests while maintaining the URI requested by the user. Unlike a nete:forward element, once the request has been redirected, the SPS is removed from the transaction, and the destination server provides resources directly to the user.

```
<nete:redirect>http://home.company.com$0</nete:redirect>
```

If a user’s request for `http://www.company.com/hr/index.html` meets a parent nete:case element’s criteria and is redirected by the above example, the user is redirected and the user’s browser displays the URL of the destination server fulfilling the request:

```
http://home.company.com/hr/index.html
```

**Note:** If you want to redirect a request over SSL, be sure to use https instead of http when defining the destination contained in the `<nete:redirect>` element.

**More information:**

[Forward and Redirect Syntax](see page 124)

---

**nete:local**

The nete:local element is included to support future functionality, and should not be included in proxy rules configuration files.
The nete:xprcond element may be used like a nete:cond element in situations where you want to apply regular expressions as conditions in your proxy rules. Regular expressions can be used to evaluate the URI string and any attached query string in your proxy rules.

The definition of the nete:xprcond element is:

```xml
<!ELEMENT nete:xprcond (nete:xpr+, nete:xpr-default)>  
```

This element must contain one or more nete:xpr elements and a single nete:xpr-default element.

A nete:xpr element is similar to a nete:cond element, and contains other elements for a rule based on a regular expression as well as a resulting behavior when the SPS finds a match for the expression. A nete:xpr-default element contains the default behavior for URI or query string combinations that do not match any of the regular expressions contained in the nete:xpr elements within a nete:xprcond element.

The definition of a nete:xpr element is:

```xml
<!ELEMENT nete:xpr (nete:rule, nete:result)>  
```

A nete:xpr element contains a nete:rule element that defines a regular expression, and a nete:result element that specifies the behavior for strings that match the regular expression.

The definition of a nete:xpr-default element is:

```xml
<!ELEMENT nete:xpr-default (nete:forward|nete:redirect|nete:local)>  
```

The nete:xpr-default element specifies a forward or redirect that should be completed if the URI or query string being evaluated by the SPS does not match the conditions stated in any of the nete:xpr elements contained in the parent nete:xprcond element.
The nete:rule and nete:result elements must be contained in a nete:xpr element. The nete:rule element specifies the regular expression that the SPS evaluates against incoming requests. The nete:result element determines the behavior for matching requests.

The definition of the nete:rule element is:

```xml
<!ELEMENT nete:rule (#PCDATA)>
```

This element contains a string that is a regular expression. The URI and query string of a request is matched against this regular expression to determine if a request satisfies the nete:xpr condition.

The definition of the nete:result element is:

```xml
<!ELEMENT nete:result (#PCDATA)>
```

nete:result elements may have the following attributes:

```xml
<!ATTLIST nete:result service (forward|redirect) "forward">
```

This element contains a string consisting of the substitution string (URL) by which the SPS creates a URL to pass to a service (forward or redirect). The service attribute is used to specify the appropriate service that will receive the URL. The default service is the forward service defined in the server.conf configuration file.

The substitution URL in the nete:result element should be a URL that optionally contains $#, where # is 0, 1, 2, etc.:

- $0 is the entire URI and query string being evaluated.
- $n is the part of the requested URI that matched the nth set of parentheses in the regular expression described in the associated nete:rule element.

For example, a set of proxy rules can contain the following:

```xml
<nete:xprcond>
  <nette:xpr>
    <nette:rule>^/realma(.*)$</nette:rule>
    <nette:result>http://server1.company.com$1</nette:result>
  </nette:xpr>
</nete:xprcond>

<nete:xpr-default>
  <nette:forward>http://www.company.com$0</nette:forward>
</nete:xpr-default>
```

Chapter 9: Configuring Proxy Rules  131
In the previous nete:xprcond proxy rule example, a request for:

http://www.company.com/realma/index.html

will be forwarded to:

http://server1.company.com/index.html

**How nete:xprcond Elements Works**

The processing of a nete:xprcond element is similar to the processing of all other nete:cond elements. As the SPS processes requests, and it encounters a nete:xprcond element in the proxy rules configuration file, the following occurs:

1. The SPS examines the first nete:xpr element in the nete:xprcond element.
2. The proxy engine evaluates the regular expression described in the nete:rule element against the URI and query string of the request.
3. If the requested URI and query string matches the regular expression specified in the nete:rule element, then the SPS resolves the result string using the results of the match, and the request is forwarded to the specified service with the URL derived from the nete:result element.
4. If the requested URI and query string do not match the regular expression in the first nete:xpr element, the proxy rules engine evaluates the next nete:xpr element (repeat steps 2 and 3). This process continues until the rules engine finds a match or reaches the nete:xpr-default element.
5. If no match is found before reaching the nete:xpr-default element, then the contents of the nete:xpr-default element determine how to route the request.

**Regular Expression Syntax**

This section describes the syntax that should be used to construct regular expressions for nete:rule elements. A nete:xprcond element takes the following form:

```xml
<nete:xprcond>
  <nete:xpr>
    <nete:rule>regular_expression</nete:rule>
    <nete:result>result</nete:result>
  </nete:xpr>
  <nete:xpr-default>forward_destination</nete:xpr-default>
</nete:xprcond>
```
In the nete:xpr element, the nete:rule element must consist of a regular expression that uses the syntax described in the following table. This syntax is consistent with the regular expression syntax supported by Apache and described at http://www.apache.org.

<table>
<thead>
<tr>
<th>Characters</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>unicode character</td>
<td>Matches any identical unicode character</td>
</tr>
<tr>
<td>\</td>
<td>Used to quote a meta-character like '/*'</td>
</tr>
<tr>
<td>\</td>
<td>Matches a single \ character</td>
</tr>
<tr>
<td>\0nnn</td>
<td>Matches a given octal character</td>
</tr>
<tr>
<td>\xhh</td>
<td>Matches a given 8-bit hexadecimal character</td>
</tr>
<tr>
<td>\uhhhh</td>
<td>Matches a given 16-bit hexadecimal character</td>
</tr>
<tr>
<td>\t</td>
<td>Matches an ASCII tab character</td>
</tr>
<tr>
<td>\n</td>
<td>Matches an ASCII newline character</td>
</tr>
<tr>
<td>\r</td>
<td>Matches an ASCII return character</td>
</tr>
<tr>
<td>\f</td>
<td>Matches an ASCII form feed character</td>
</tr>
<tr>
<td>[abc]</td>
<td>Simple character class</td>
</tr>
<tr>
<td>[a-zA-Z]</td>
<td>Character class with ranges</td>
</tr>
<tr>
<td>^abc</td>
<td>Negated character class</td>
</tr>
<tr>
<td>[:alnum:]</td>
<td>Alphanumeric characters</td>
</tr>
<tr>
<td>[:alpha:]</td>
<td>Alphabetic characters</td>
</tr>
<tr>
<td>[:blank:]</td>
<td>Space and tab characters</td>
</tr>
<tr>
<td>[:cntrl:]</td>
<td>Control characters</td>
</tr>
<tr>
<td>[:digit:]</td>
<td>Numeric characters</td>
</tr>
<tr>
<td>[:graph:]</td>
<td>Characters that are printable and are also visible (A space is printable, but not visible, while an ‘a’ is both)</td>
</tr>
<tr>
<td>[:lower:]</td>
<td>Lower-case alphabetic characters</td>
</tr>
<tr>
<td>[:print:]</td>
<td>Printable characters (characters that are not control characters)</td>
</tr>
<tr>
<td>[:punct:]</td>
<td>Punctuation characters (characters that are not letter, digits, control characters, or space characters)</td>
</tr>
<tr>
<td>[:space:]</td>
<td>Space characters (such as space, tab, and formfeed)</td>
</tr>
<tr>
<td>[:upper:]</td>
<td>Upper-case alphabetic characters</td>
</tr>
<tr>
<td>[:xdigit:]</td>
<td>Characters that are hexadecimal digits</td>
</tr>
<tr>
<td>Characters</td>
<td>Results</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>[:javastart:]</td>
<td>Start of a Java identifier</td>
</tr>
<tr>
<td>[:javapart:]</td>
<td>Part of a Java identifier</td>
</tr>
<tr>
<td>.</td>
<td>Matches any character other than newline</td>
</tr>
<tr>
<td>\w</td>
<td>Matches a &quot;word&quot; character (alphanumeric plus &quot;)&quot;)</td>
</tr>
<tr>
<td>\W</td>
<td>Matches a non-word character</td>
</tr>
<tr>
<td>\s</td>
<td>Matches a whitespace character</td>
</tr>
<tr>
<td>\S</td>
<td>Matches a non-whitespace character</td>
</tr>
<tr>
<td>\d</td>
<td>Matches a digit character</td>
</tr>
<tr>
<td>\D</td>
<td>Matches a non-digit character</td>
</tr>
<tr>
<td>^</td>
<td>Matches only at the beginning of a line</td>
</tr>
<tr>
<td>$</td>
<td>Matches only at the end of a line</td>
</tr>
<tr>
<td>\b</td>
<td>Matches only at a word boundary</td>
</tr>
<tr>
<td>\B</td>
<td>Matches only at a non-word boundary</td>
</tr>
<tr>
<td>A*</td>
<td>Matches A 0 or more times (greedy)</td>
</tr>
<tr>
<td>A+</td>
<td>Matches A 1 or more times (greedy)</td>
</tr>
<tr>
<td>A?</td>
<td>Matches A 1 or 0 times (greedy)</td>
</tr>
<tr>
<td>A{n}</td>
<td>Matches A exactly n times (greedy)</td>
</tr>
<tr>
<td>A{n,m}</td>
<td>Matches A at least n but not more than m times (greedy)</td>
</tr>
<tr>
<td>A*?</td>
<td>Matches A 0 or more times (reluctant)</td>
</tr>
<tr>
<td>A+?</td>
<td>Matches A 1 or more times (reluctant)</td>
</tr>
</tbody>
</table>
### Regular Expression Syntax

<table>
<thead>
<tr>
<th>Characters</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>A??</td>
<td>Matches A 0 or 1 times (reluctant)</td>
</tr>
<tr>
<td>AB</td>
<td>Matches A followed by B</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>(A)</td>
<td>Used for subexpression grouping</td>
</tr>
<tr>
<td>\1</td>
<td>Backreference to 1st parenthesized subexpression</td>
</tr>
<tr>
<td>\n</td>
<td>Backreference to nth parenthesized subexpression</td>
</tr>
</tbody>
</table>

All closure operators (+, *, ?, {m,n}) are greedy by default, meaning that they match as many elements of the string as possible without causing the overall match to fail. If you want a closure to be reluctant (non-greedy), you can simply follow it with a ‘?’$. A reluctant closure will match as few elements of the string as possible when finding matches. {m,n} closures don’t currently support reluctancy.

### Regular Expression Examples in nete:rule and nete:result

Regular expressions offer a very flexible and powerful tool that can be employed in SPS proxy rules. This section provides a few examples nete:rule elements in proxy rules. In addition, the examples also contain various uses of the nete:result element to show how groupings in a nete:rule can be used to affect the destination generated by the children of a nete:xprcond element.

#### Map Single Rule to Many Destination Servers

In the following example, a nete:rule element contains a regular expression that can be used to forward requests to many different destinations. This example assumes that the SPS will receive URIs that take the following form:

/GOTO=some path and or filename

Consider a nete:xprcond element contains the following child elements:

```xml
<nete:xpr>
    <nete:rule>/GOTO=(.*)/(.*)</nete:rule>
    <nete:result>http://$1/$2</nete:result>
</nete:xpr>
```
The regular expression in the nete:rule element and the associated nete:result element match URIs that produce a /GOTO=string. Upon finding a match, the SPS uses the first string after the = symbol in the URI as the $1 value of the result, and the value following the first / symbol that appears after the = symbol as the $2 result. The nete:result element combines these elements to create a URL. By default, nete:result elements use the SPS forward service.

For example, if the URI of a request evaluated by the nete:xpr element described above were as follows:

/GOTO=server1.company.com/index.html

Then the regular expression in the nete:rule element would find a match and assign the value of $1 as server1.company.com and the value of $2 as index.html. The nete:result element assembles these values into the following URL:

http://server1.company.com/index.html

This URL is the target which the SPS uses to resolve the request.

**Regular Expressions to Redirect Users**

The nete:result element can also be used to create a redirect response that is returned to the user requesting the resource. This forces the fulfillment of a request to be handled by a server other the SPS after authentication and authorization. The following is an example of a nete:xpr element that specifies a redirect in the nete:result child element.

```
<nete:xpr>
    <nete:rule>/REDIR=(.*)/(.*)</nete:rule>
    <nete:result service="redirect">http://$1/$2</nete:result>
</nete:xpr>
```

**Note:** The service attribute instructs the SPS to use the redirect service in place of the default forward service.

**Header Values in Forwards, Redirects, and Results Filters**

The value of an HTTP header or a SiteMinder response header can be substituted directly into a nete:forward, nete:redirect, or nete:result element. When a URI in a forward or redirect element, or a rule in a result filter element contains `{{HEADER_NAME}}`, the proxy engine searches for a header in a request that matches the specified header and substitute's the header value before resolving the forward, redirect, or result. If no matching header is found in a request, the proxy engine substitutes an empty string in place of the header value.

**Note:** Header names are case sensitive.
Dynamic Header Value in a nete:forward

To use a dynamic header value as part of a nete:forward element, simply insert \{{HEADER_NAME}\} into the URL portion of the forward. For example:

```
<nete:forward>http://www.company.com/{{RESPONSE1}}$1</nete:forward>
```

You can use multiple headers in a single nete:forward element. For example:

```
<nete:forward>http://www.company.com/{{RESPONSE1}}/{{RESPONSE2}}$1</nete:forward>
```

Dynamic Header Value in a nete:redirect

To use a dynamic header value as part of a nete:redirect element, simply insert \{{HEADER_NAME}\} into the URL portion of the redirect. For example:

```
```

You can use multiple headers in a single nete:redirect element. For example:

```
```

Dynamic Header Value in a nete:result

To use a dynamic header value as part of a nete:result element, simply insert \{{HEADER_NAME}\} into the URL portion of the result. For example:

```
<nete:result>http://www.company.com/{{HEADER_NAME}}$1</nete:result>
```

You can use other features of proxy rules, such as filters, in conjunction with a dynamic header value. For example:

```
<nete:result filter="filter1">http://$1/$2?{{HEADER_NAME}}</nete:result>
```
Response Handling

The SPS uses SiteMinder responses to determine a destination for a request. Since transactions that are routed through the SPS include an interaction between the SPS web agent and SiteMinder, any SiteMinder responses gathered during the authentication and authorization process may be used by the SPS to determine the destination of a request.

For example, if a user directory contains information about the account type for a banking web site, the SPS can proxy users with different types of accounts to different destinations. This enables an enterprise to provide a higher quality of service to its best customers. Customers with standard accounts can be handled by one set of destination servers, while customers with premium accounts can be handled by a separate set of high performance destination servers.

Modify Proxy Rules

To modify proxy rules you must edit the proxy rules XML configuration file using a text editor. Since proxy rules are XML files, your proxy rules configuration file must be well-formed and valid. Remember that the tags in a well-formed XML file must all consist of opening and closing tags. To be valid, the file must adhere to the guidelines laid out in the proxyrules.dtd.

Changes to the proxy rules XML configuration file are picked up automatically by the SPS. When the SPS receives a request, it checks whether or not the proxy rules have changed. If the file has changed, the rules are reloaded before fulfilling the request.

Note: If you change the name of the proxy rules XML configuration file in the rules_file directive in the <ServiceDispatcher> element of the server.conf file, you must restart the SPS.

Sample Proxy Rules Configuration Files

The SPS installs several examples of proxy rules configuration files. You can use these example XML files as the basis for your own proxy rules files.

You can find these example files in the directory `sps_home\secure-proxy\proxy-engine\examples\proxyrules`. We recommend you look at the example file as you are reading the descriptions in this guide.

You may copy and customize a file to suit your needs.
To customize and deploy a proxy rules file

1. Navigate to the directory
   \spshome\secure-proxy\proxy-engine\examples\proxyrules.
2. Make a copy of the example file you want to use.
3. Customize the content and save the new file under a unique name.
4. Copy the modified file to the directory \spshome\secure-proxy\proxy-engine\conf.
5. Open up the server.conf file to modify the proxy rules section of the file to point to the customized file.

Proxy Rules Example—Routing Requests by Virtual Host

The example file proxyrules_example1.xml file routes requests based on the hostname specified in the request.

In this file, a simple set of proxy rules routes user requests based on the virtual host specified in the requested resource. All requests to the bondtrading.company.com server are forwarded to server2, all requests to banking.company.com are forwarded to server1, and all other requests are forwarded to the companies home server, which is the default for requests that do not match the criteria in any other nete:cond element.

Note: The nete:case elements must specify a port, since the port number is considered as part of the virtual host requested by the user. Use the beginswith criteria to avoid needing port numbers.

The following table illustrates the results of requests using the proxy rules based on virtual hosts.

<table>
<thead>
<tr>
<th>Requested URL</th>
<th>Forwarded URL</th>
</tr>
</thead>
</table>
Proxy Rules Example—Routing Requests by Header Value

The example file proxyrules_example2.xml file routes SPS requests based on the value of an HTTP header. The HTTP header can be a standard header or one created using a SiteMinder response.

**Note:** For information about SiteMinder responses, see the **CA SiteMinder Policy Design**.

In this example, assume that the SPS routes requests made to a default virtual host of www.company.com.

In this file, the value of the HTTP header variable "HEADER" determines the destination for the request.

The following table illustrates the results of requests using the proxy rules based on an HTTP header.

<table>
<thead>
<tr>
<th>Requested URL</th>
<th>Forwarded URL</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.company.com/index.html">http://www.company.com/index.html</a> HTTP_HEADER has the following value: HTTP_HEADER=&quot;value1&quot;</td>
<td><a href="http://server1.company.com/index.html">http://server1.company.com/index.html</a></td>
</tr>
<tr>
<td><a href="http://www.company.com/index.html">http://www.company.com/index.html</a> HTTP_HEADER has the following value: HTTP_HEADER=&quot;value2&quot;</td>
<td><a href="http://server2.company.com/index.html">http://server2.company.com/index.html</a></td>
</tr>
<tr>
<td><a href="http://www.company.com/index.html">http://www.company.com/index.html</a> HTTP_HEADER has a value other than value1 or value2.</td>
<td><a href="http://home.company.com/index.html">http://home.company.com/index.html</a></td>
</tr>
</tbody>
</table>

**Note:** You do not need to include the HTTP_ of the header variable name in the nete:cond element. SPS assumes HTTP_ for header variable names.

Proxy rules that use header values are an excellent way to forward requests based on a desired level of service. For example, you can use the value of an HTTP header variable that contains a user account types to distribute requests to high performance servers for customers with premium accounts.
Proxy Rules Example—Routing Requests by Device Type

The example file proxyrules_example3.xml file routes SPS requests based on the type of device used to access the resource.

**Note:** The user-agent HTTP header value is used to determine how to route requests.

In the file, users who access resources using a browser (user agent contains Mozilla for Web browsers) are forwarded to a Web server, while all other users are forwarded to a wireless server.

The following table illustrates the results of requests using the proxy rules based on a device type.

<table>
<thead>
<tr>
<th>Requested URL</th>
<th>Forwarded URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>User access resource via a Web browser.</td>
<td></td>
</tr>
<tr>
<td>User access resource via a wireless device.</td>
<td></td>
</tr>
</tbody>
</table>

Proxy Rules Example—Routing Requests with URIs

The example file proxyrules_example4.xml file routes SPS requests based on the URI specified in the user request.

The following table illustrates the results of requests using the proxy rules based on URIs.

<table>
<thead>
<tr>
<th>Requested URL</th>
<th>Forwarded URL</th>
</tr>
</thead>
</table>
Proxy Rules Example—Routing Requests by File Extension

The example file proxyrules_example5.xml file routes SPS requests based on the file extension requested by the user. This is achieved by using the URI condition in combination with the endswith criteria.

In the file, the `<nete:forward>` and `</nete:forward>` tags appear on separate lines due to space constraints. However, in your proxy rules configuration files, the opening and closing tags for a `<nete:forward>` element must appear on the same line. If they do not, the SPS interprets the line break as part of the forward URL, which causes requests to be forwarded incorrectly.

In the previous example, users who access .jsp resources are forwarded to an application server, while wireless users are forwarded to the wireless server. All other users are forwarded to the home server.

The following table illustrates the results of requests using the proxy rules based on file extensions.

<table>
<thead>
<tr>
<th>Requested URL</th>
<th>Forwarded URL</th>
</tr>
</thead>
</table>

Proxy Rules Example—Routing Requests with Nested Conditions

The example file proxyrules_example6.xml file routes SPS requests based on the hostname, certain headers, and device types. This file demonstrates how the SPS can handle complex relationships in a single configuration file.

In the file, the `<nete:forward>` and `</nete:forward>` elements must be located on the same line. In the example, the `</nete:forward>` closing tags sometimes appear on a separate line due to space constraints, however, a line break in an actual proxy rules file causes an error. The SPS interprets line breaks before the `</nete:forward>` closing tag as characters that are part of the URL contained in the nete:forward element.
The following table illustrates the results of requests using proxy rules with nested conditions.

<table>
<thead>
<tr>
<th>Requested URL</th>
<th>Forwarded URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>with a header value of GOLD_USER=&quot;yes&quot;</td>
<td></td>
</tr>
<tr>
<td>with a header value of GOLD_USER=&quot;no&quot;</td>
<td></td>
</tr>
<tr>
<td>with a USER_AGENT header value that contains a</td>
<td></td>
</tr>
<tr>
<td>wireless device name</td>
<td></td>
</tr>
<tr>
<td>with a USER_AGENT header value that does not</td>
<td></td>
</tr>
<tr>
<td>contains a wireless device name</td>
<td></td>
</tr>
</tbody>
</table>

**Proxy Rules Example—Using Regular Expression in Proxy Rules**

The example file proxyrules_example7.xml file routes SPS requests based on nete:xprcond elements that contain regular expressions. Regular expressions are evaluated based on the URI and query string of a request.

In the file, the URI and query string of the request are evaluated against the three regular expressions defined in the nete:xpr elements. If a match is not found against the first nete:xpr element, the SPS tries to match it against the second, and finally the third regular expression. If no matches are found, the nete:xpr-default condition is used to handle the request.

The following table lists the results of requests using the regular expression proxy rules.

<table>
<thead>
<tr>
<th>Requested URL</th>
<th>Forwarded URL</th>
</tr>
</thead>
</table>
### Requested URL vs. Forwarded URL

<table>
<thead>
<tr>
<th>Requested URL</th>
<th>Forwarded URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>User is redirected so that server2.company.com must directly fulfill the user’s request.</td>
<td></td>
</tr>
</tbody>
</table>
This section contains the following topics:

SPS Deployment in an Enterprise (see page 145)

SPS Deployment in an Enterprise

The SPS uses reverse proxy architecture to enable access control, single sign-on, and SSL acceleration. It does not provide the content caching and some other features provided by traditional reverse proxy servers. The SPS is intended to be an addition to enterprise architecture, rather than a replacement for other proxy technologies. As such, the SPS can be placed in clusters with load balancing devices and caching devices on either side of the clusters.

The following illustration shows how the SPS can be inserted into a network to work in conjunction with load balancing devices.

Note: In addition to load balancing devices, caching devices can be placed on either side of the SPS cluster.
Sticky-Bit Load Balancing

When using the cookieless session schemes supported by the SPS, session information for users who access resources through SPS is maintained in an in-memory session store. Because the session information is maintained at the SPS where a user is first authenticated, the same SPS should be used for all the user requests in a single session. When implemented in clusters, the SPS must be used in conjunction with sticky-bit load balancers to provide a consistent connection to the same SPS, enabling single sign-on when using session schemes other than the traditional SiteMinder cookie session scheme.

To deploy the SPS using cookieless session schemes the following must be considered:

- In most deployments, the SPS is deployed in clusters, with several servers sharing the load of incoming requests. The load balancing is handled by load balancer devices. These devices must have sticky bit capability to maintain single sign-on.

  Sticky bit load balancers ensure that once a user’s session is established with a specific SPS in a cluster, that SPS services all of the user’s requests. This capability is required because the SPS maintains session information for cookie-less sessions in active memory. If a user’s request is not handled using sticky bit technology, the user will be charged for new credentials each time a request is fulfilled by a different SPS in the cluster of servers.

- When configuring the settings for the SPS, the default virtual host defined in the server.conf file of the SPS must be defined using the name and IP address of the load balancing device.

- The load balancing device must be configured as the point of entry to the SPS.

- The load balancing device must point to the cluster of SPSs.

- The httpd.conf file, located in the sps_home/secure-proxy/httpd/conf directory, must be modified so that the value of the ServerName directive is set to the name of the load balancing device, not the system on which you installed the SPS.

- If using SSL, a certificate must be issued to the load balancer, not the SPS.

- The system or systems on which you install the SPS must have approximately 1K of memory for each simultaneous user session that will be maintained in the in-memory session store. For example, if a single system must maintain 1000 concurrent sessions, the system must have 1 megabyte of RAM available for this purpose.
Proxying to Trusted Sites vs. Non-Trusted Sites

The SPS is assumed to proxy for trusted sites within the enterprise. As part of a proxy transaction, SiteMinder generated HTTP header variables and any variables generated by SiteMinder responses are forwarded along with each HTTP and HTTPS request. These responses can be used by other enterprise applications.

Important! If you employ the SPS in transactions that proxy for content on non-trusted sites, the headers that accompany the transaction will also be forwarded to the non-trusted sites. We recommend using the SPS to proxy for destination servers trusted by your enterprise.

Configuring Virtual Hosts for the SPS

You can configure the SPS with multiple hosts and act as a virtual host for one or more hostnames.

To configure SPS as multiple hosts and act as a virtual host for one or more hostnames

1. Edit the <VirtualHost> parameters of the server.conf file to configure the SPS to act as a virtual host for one or more hostnames.
2. Edit the configuration file for the embedded Apache Web server.

More information:

Virtual Host Names Configuration (see page 114)

Edit the Apache Configuration File To Handle Multiple Virtual Hosts

When you are running multiple virtual hosts in the same operating environment with the SPS, and transactions run in this environment, update the Apache configuration file (httpd.conf). This file is located in the directory \sps_home\secure-proxy\httpd\conf. If SSL is enabled for the web server, also make the same updates to the httpd-ssl.conf file, which is located in the \sps_home\secure-proxy\httpd\conf\extra directory. The updates vary depending on whether your operating environment is based on IPv4 or IPv6.
To update the httpd.conf file, and optionally the httpd-ssl.conf file, to handle multiple virtual hosts

- For IPv4 environments, add the following LISTEN directive:
  
  LISTEN 127.0.0.1:<_port>

- For IPv6 environments, add the following LISTEN directive:
  
  LISTEN [::1]:<_port>

- For dual stack environments with IPv4 and IPv6 supports, add the following LISTEN directives:
  
  LISTEN 127.0.0.1:<_port>

  LISTEN [::1]:<_port>

In addition, update the loopback address entry in the hosts file so that the new host name is added, as follows:

- IPV4: 127.0.0.1

- IPV6: [::1]

The hosts file is usually located on Windows in C:\WINDOWS\system32\drivers\hosts.
On UNIX, the hosts file is usually in /etc/hosts.

**Implementing Session Scheme Mappings for Multiple Virtual Hosts**

If you want to configure the SPS to recognize multiple user agent types, assign different session scheme mappings for those user agents based on virtual hosts, you must follow these steps:

1. Configure session schemes, or verify the configuration of the schemes included with the SPS.
2. Define user agent types in the server.conf file.
3. Create a section for each virtual host in the server.conf file that defines any directives that differ from default settings (refer to Overriding Default Values for a Virtual Host).
4. Define session scheme mappings for each virtual host.
The following excerpts from a server.conf file provide an example where a user agent type has been defined for Internet Explorer (IE) browser users. IE users will be mapped to use session schemes other than the default session scheme defined for a virtual host. The following example shows the session schemes defined in the server.conf file.

```xml
# Session Schemes
<SessionScheme name="default">
    class="com.netegrity.proxy.session.SessionCookieScheme"
    accepts_smsession_cookies="true"
</SessionScheme>

<SessionScheme name="ssl_id">
    class="com.netegrity.proxy.session.SSLIdSessionScheme"
    accepts_smsession_cookies="false"
</SessionScheme>

<SessionScheme name="simple_url">
    class="com.netegrity.proxy.session.SimpleURLSessionScheme"
    accepts_smsession_cookies="false"
</SessionScheme>

<SessionScheme name="minicookie">
    class="com.netegrity.proxy.session.MiniCookieSessionScheme"
    accepts_smsession_cookies="false"
    cookie_name="MiniMe"
</SessionScheme>
```

The following example shows the definition of the IE user agent type. This user agent type will be referenced when defining session scheme mappings later in the server.conf file.

```xml
# TO-DO: Define Any User Agents, if you want to
# use a different session scheme based on
# the type of client accessing the server.
#
# NOTE:  UserAgent matching is done in the order
# in which the user agents are defined in this file.
<UserAgent name="IE">
    User-Agent="MSIE"
</UserAgent>

# <UserAgent name="NS">
#     User-Agent=some other regular expression
# </UserAgent>
```

The preceding example shows that the default session scheme specified in the defaultsessionscheme directive is mini-cookie. This session scheme will be used for all transactions unless another session scheme is explicitly included in a session scheme mapping, or another scheme overrides the default session scheme in the definition of a virtual host.
The `<VirtualHostDefaults>` directive shows the session scheme mapping for the IE user agent type that was defined in `<UserAgent name="IE">`. This mapping indicates that for all virtual hosts using default session scheme mappings, IE browser users' sessions will be maintained using the simple URL rewriting sessions scheme.

```xml
<VirtualHostDefaults>
    # Service Dispatcher
    <ServiceDispatcher>
        class="com.netegrity.proxy.service.SmProxyRules"
        rules_file="conf\proxyrules.xml"
    </ServiceDispatcher>
    # default session scheme
    defaultsessionscheme="minicookie"
    #TO-DO: Define any session scheme mappings
    <SessionSchemeMappings>
        # user_agent_name=session_scheme_name
        IE="simple_url"
        # NS=simple_url
    </SessionSchemeMappings>
</VirtualHostDefaults>
```

The Virtual Host directives show the server name and IP address for the default virtual host configured for the SPS.

```xml
# Default Virtual Host
<VirtualHost name="default">
    hostnames="server1, server1.company.com"
    addresses="192.168.1.10"
    # The defaults can be overridden
    # not only for the Virtual Host
    # but for the WebAgent for that virtual host as well
    #<WebAgent>
    #</WebAgent>
</VirtualHost>
```
The Virtual Host directive for additional virtual host shows the specific default virtual host settings that will be overridden for the server2 virtual host. Notice that these overrides include new session scheme mappings. The default scheme for server2 is default. In Session Scheme directive the default is defined as the traditional SiteMinder cookies session scheme. Further, the session scheme mapping for IE users in Virtual Host directives is also mapped to the default scheme. Therefore, the SPS will use SiteMinder cookies session scheme to maintain sessions for all users who access server2.

```xml
# Additional Virtual Host  
<VirtualHost name="host2">  
  requestblocksize="4"  
  responseblocksize="4"  
  hostnames="server2, server2.company.com"  
  #addresses="192.168.1.15"  
  # default session scheme  
  defaultsessionscheme="default"  
  #TO-DO: Define any session scheme mappings  
  <SessionSchemeMappings>  
    #user_agent_name=session_scheme_name  
    IE="default"  
  </SessionSchemeMappings>  
  #<WebAgent>  
  #</WebAgent>  
</VirtualHost>
```
Chapter 11: Integrating the SPS with SiteMinder

This section contains the following topics:

- How the SPS Interacts with SiteMinder (see page 153)
- SPS and SharePoint Resources (see page 159)
- SPS and ERP Resources (see page 159)
- Password Services for SPS (see page 160)
- Configuring Managed Self Registration for the SPS (see page 162)
- Firewall Considerations (see page 165)
- HTTP Header Configuration for Sun Java Web Servers (see page 165)
- HTTP Header for SiteMinder Processing with SPS (see page 166)
- Handling Encoded URLs (see page 166)

How the SPS Interacts with SiteMinder

SiteMinder is a solution for securely managing e-business. SiteMinder consists of a Policy Server that allows you to specify policies for your enterprise, and Web Agents that are installed on web servers. The Web Agents communicate with the Policy Server to provide authentication, authorization, and other functions.

The SPS contains a Web Agent that is compatible with the SiteMinder Web Agent and Policy Server technology. As all SiteMinder Web Agents, the SPS must be configured as an object in SiteMinder. Policies must be created that determine authentication and authorization requirements for accessing destination servers.

SiteMinder objects are configured using SiteMinder <adminui>. You can configure the following objects:

Agent

Configure an agent object with settings for the Web Agent included in the SPS. Specify this Web Agent when creating realms.

User Directories

Configure connections to any user directories that authenticate and authorize users.

Policy Domains

Configure policy domains that contain realms, rules, and policies.
How the SPS Interacts with SiteMinder

Realms
Configure realms that contain resources you want to protect with SiteMinder.

Rules
Configure rules that identify specific resources and actions that you want to protect with SiteMinder.

Responses
Configure any responses that can return information to applications, or to the SPS. Information returned to the SPS can determine how to route user requests.

Policies
Configure policies that bind users and groups to rules and responses.

Note: For complete information about how to configure SiteMinder objects, see the CA SiteMinder Configuration Guide.

Authentication Scheme Considerations

SiteMinder enforces authentication schemes to protect resources. When users attempt to access protected resources through a SiteMinder Web Agent or the SPS, SiteMinder asks for credentials based on the authentication scheme protecting the resource.

SiteMinder also provides protection levels to each authentication scheme. The protection levels are enforced during single sign-on when the user tries to access resources protected by different authentication schemes. In such scenarios, the users can access resources protected by different authentication schemes without reauthentication if the protection levels for each of the authentication schemes are equal or lower. When moving from a lower protection level to a higher protection level, the user is challenged for authentication. When moving from a higher protection level to a lower protection level, the user is not challenged for reauthentication.

When the SPS is integrated with SiteMinder, the SPS behaves similar to a SiteMinder Web Agent. However, SPS using basics authentication behaves similar to a Web Agent only if the SPS is configured to use default SessionCookieScheme scheme to track user sessions. If the SPS is configured to use any of the other advanced or cookieless session schemes, the user has to reauthenticate. Single sign-on does not work.

For example, a basic authentication scheme with a protection level of 5 protects two resources, resource1 and resource2. The SPS is configured to use a mini-cookie session scheme (or any other cookieless session scheme) to track user sessions. When a user tries to access resource1, the SPS forwards the request to SiteMinder. SiteMinder verifies the authentication scheme for resource1 and challenges the user for credentials.
The SPS collects the credentials from the user and after successful authentication by SiteMinder, allows the user to access resource1. If the user then tries to access resource2, the SPS forwards the request to SiteMinder. SiteMinder verifies the authentication scheme for resource2 and challenges the user for credentials. Because the SPS is configured to use mini-cookie session scheme, the SPS requests the user to reauthenticate. If the SPS is configured to use the default SiteMinder cookie session scheme, then the user need not reauthenticate to access resource2.

**Note:** For more information about authentication schemes and their protection levels, see the *CA SiteMinder Policy Configuration Guide*.

### Proxy-Specific WebAgent.conf Settings

A number of settings in the WebAgent.conf configuration files for Web Agents installed behind the DMZ in an enterprise have specific implications for the SPS.

The settings that must be modified in the destination server WebAgent.conf files include:

**proxytrust**

As an optimization, the proxytrust directive can be set on the destination server Web Agent sitting behind the SPS. Enter one of the following settings:

- **yes**
  
  The destination server web agent automatically trusts the authorizations made by the SPS.

- **no**
  
  The destination server web agent requests authentication every time. (Default)

**proxytimeout**

Instructs the Web Agent on the destination server to time out the single sign-on token used in requests made by the SPS. Enter a value in seconds.

**Default:** 120 seconds.
How the SPS Interacts with SiteMinder

Avoiding Policy Conflicts with Destination Server Web Agents

In some deployments, when the SPS is running in proxy trust mode, the SPS can protect resources from one set of users, while a Web Agent on a destination server protects the same resources from another set of users.

In the following illustration, Destination Server 2 has its own Web Agent. Extranet users are authenticated and authorized at the SPS, while Intranet users are authenticated and authorized through the Web Agent on the destination server. In such situations policies must exist in SiteMinder policy store for the embedded SPS Web Agent and the Web Agent on the destination server.

Note: When creating policies, administrators must be sure that the policies do not conflict with each other. If policies contradict one another, it is possible that SiteMinder may allow unwanted or unexpected behavior.

To correctly create policies and other required SiteMinder objects for the resources contained on Destination Server 2, the following objects could be created in SiteMinder:

- SPS Web Agent
- Destination server 2 Web Agent
- Realm using the SPS Web Agent
- Realm using the destination server 2 Web Agent
- Rule for resources accessed through the SPS Web Agent
- Rule for resources accessed through the destination server 2 Web Agent
Policy for SPS web agent resources
- Policy for destination server 2 Web Agent resources

The following illustration shows how two policies must be created to protect a single resource when Compatibility Mode is used in environments that include both the SPS and Web Agents.

In the preceding illustration, the rules and realms for the same resources may have different paths based on the location of the resources on the destination server and the proxy rules used to forward requests.

For example, using the sample configuration in the preceding illustration, a resource called banking.html may be located on Destination Server 2 in the server2.company.com/start/banking/ directory, but the SPS may have proxy rules that forward all requests for www.company.com/banking/banking.html to the same destination on Server 2. Therefore, the same resource can have two different SiteMinder rules that represent the same resource. One rule authorizes access to the resource directly for employees on the intranet, and the other authorizes employees on the road who want to access the same resource from the extranet.
Configuring SiteMinder Rules that Redirect Users

SiteMinder provides the ability to create response objects that redirect a request under certain conditions. For example, you can create a response that redirects a request to a custom error page after a failed authentication (OnRejectRedirect). By default for cookieless session schemes that rewrite a requested URL (Simple URL Rewriting), the SPS recognizes a user session information after a redirection.

To terminate a user session after a redirection, create a response attribute in SiteMinder for the relevant policy that modifies the value of the SiteMinder SM_REWRITE_URL header. This HTTP header must be set to NO to force a new session after a redirection.

For example, if you have a resource in realmA that is protected by an authentication scheme with a protection level of 5, and a second resource in realmB protected by authentication scheme with a protection level of 10, a user who successfully authenticates in realmA will be challenged for credentials when moving to realmB (due to the higher protection level).

If an OnRejectRedirect response is associated with realmB and the user fails to authenticate when challenged for credentials in realmB, the default behavior of the SPS maintains the user’s original session information even after the user is redirected to a custom error page.

To terminate the user’s session after the redirect and force an entirely fresh session on the next login attempt, you must create a response attribute that sets the SM_REWRITE_URL=NO, and associate the response with the appropriate policy.
SPS and SharePoint Resources

If you want to use the SPS to secure resources managed by Microsoft SharePoint, make the configuration changes listed following.

- In the SPS Agent Configuration Object, set the following parameters:
  - SPClientIntegration = server_name:port
    - The server name must match the value set for the ServerName field in the httpd.conf file. Most often, the ServerName is a fully qualified host name, but the value can be an IP address.
  - ProxyAgent = Yes

  **Note:** These parameters can also be added to the SPS LocalConfig.conf file.

- In the SPS WebAgent.conf file, add a LoadPlugin parameter that points to the location SharePoint plugin (SPPlugin.dll on Windows, libSPPlugin.so on Solaris).

- In the server.conf file, add a VirtualHost element with the following parameters:
  ```
  <VirtualHost name="VHName">
    hostnames="host_name, host_name"
    enableredirectwrite="yes"
    redirectwritablehostnames="server1.company.com, domain1.com"
  </VirtualHost>
  ```

  **Note:** For more information, see the CA SiteMinder Agent for SharePoint Guide.

SPS and ERP Resources

You can use the SPS to secure resources managed by an ERP system. The SPS can function as a proxy in front of ERP agents protecting the following ERP systems:

- Siebel Application Server
- PeopleSoft Application Server
- SAP AS Web Application Server
- SAP ITS Application Server

While the ERP agent must be installed on the ERP server, the SPS secures the ERP resources at the Policy Server.

**Note:** For information about the Policy Server settings required to support the ERP server, see the appropriate CA ERP agent guide.
To configure the SPS as a reverse proxy for an ERP agent

1. Specify the ERP server and appropriate port number for the `<nete:forward>` element in the proxyRules.xml file.

2. Specify the following values in the server.conf file:
   - Set the value of the enableredirectrewrite parameter to "yes".
   - Set the value of the redirectrewritablehostnames parameter to the host name of the system on which the ERP server is running. For example:
     ```xml
     <VirtualHost name="sales">
       hostnames="sales, sales.company.com"
       enableredirectrewrite="yes"
       redirectrewritablehostnames="server1.company.com,domain1.com"
     </VirtualHost>
     ```
   - Set the value of the addquotestocookie parameter in the `<Server>` section of the server.conf to "no". For example:
     ```
     addquotestocookie="no"
     ```

   **Note:** For information about any required settings on the ERP Server side, refer to the documentation for your ERP server.

The SPS is configured as a proxy for the ERP agent.

---

**Password Services for SPS**

Password services are a SiteMinder feature that provides an additional layer of security to protected resources by allowing a SiteMinder administrator to manage user passwords. Password services allow an administrator to create password policies that define rules and restrictions governing password expiration, composition, and usage.

When configuring password services in SiteMinder, a password policy is associated with a directory. All users contained in the directory, or some part of the directory identified by an LDAP search expression, must adhere to the password policy. Password services are processed from inside the Apache Web server rather than from a back-end Web server hosting an agent.

**Note:** For more information about password services, see the *Policy Design Guide*. 

---
Configure a Password Policy for SPS

For SiteMinder to implement Password Services in a SPS deployment, the redirection URL specified in the Policy Server User Interface must refer to the SPS server, with the addition of a specific virtual directory path.

**To configure a password policy for SPS**

1. Log in to the Policy Server User Interface.
3. Click the User Directories object.
4. In the User Directory List select the user directory you want to protect with Password Services.
5. Right click and select Properties of User Directory.
   The User Directory Properties dialog appears.
6. In the Credentials and Connection tab, select Require Credentials.
7. Enter the administrator's credentials, including the Username and Password.
8. In the User Attributes tab of the same dialog, enter names for the following directory user profile attributes:
   - Universal ID (example: uid)
   - Disabled Flag (example: carLicense)
   - Password Attribute (example: userPassword)
   - Password Data (example: audio)
   **Note:** For more information on the User Directory Properties dialog, see the Policy Server User Interface help.
9. Click OK.
10. In the System tab, select the Password Policies object.
11. Right click on the Password Policies object and select Create Password Policy.
    The Password Policy Properties dialog appears.
12. In the General tab, select the name of the user directory for which you made the settings for Password Services.
13. In the General tab, specify a Redirection URL as follows:
    /siteminderagent/pw/smpwservicescgi.exe
14. Click OK.
    The configuration is complete.
Verify Password Services for SPS

After you have configured Password Services for the SPS, you can perform a simple test to see whether Password Services are in effect.

To verify whether Password Services is working
1. Select the password-protected directory from the User Directory list.
2. Select Manage User Accounts from the Tools menu.
   The User Management dialog appears.
3. Select a user.
4. Select "User must change password at next login."
5. Click OK.

When you next request a protected page in SPS and are challenged, enter the credentials for the specified user. The Password Change screen appears, indicating the Password Services is working.

Configuring Managed Self Registration for the SPS

Managed Self-Registration (MSR) is a SiteMinder feature that allows users to login to a web site and establish a new user account. New users can access the web site, provide personal information, and receive an account on the site. In addition, users can also define a hint that may be used to retrieve a forgotten password.

When configuring MSR in SiteMinder, a registration scheme must be configured. This registration scheme can be specified in the realm that contains the registration URL. The registration realm requires an HTML forms-based authentication scheme that points to an appropriate template for MSR services.
The following illustration shows the user of MSR with SPS.

![Diagram showing the user of MSR with SPS](image)

The additional web server, which should reside outside of the DMZ, must contain a SiteMinder Web Agent installation. This web agent installation provides the processing required by MSR, and ensures that the processing of sensitive data occurs behind the DMZ.

**Install a Web Agent for MSR**

To use MSR with the SPS, you must install a SiteMinder Web Agent on a Web server behind the DMZ. For information about installing a Web Agent, see the *CA SiteMinder Web Agent Installation Guide*.

After the Web Agent is installed, note the following:

- The Web Agent does not need to be enabled.
- The Web Agent must be configured as an agent object in SiteMinder.
- The Web Agent must be a version 6.x Web Agent; r12 Web Agents do not support MSR.

The SPS uses the MSR servlet configured for the web agent. The web agent is not used for authentication or authorization.
Configure the Policy Server User Interface for MSR

For SiteMinder to properly implement MSR in a SPS deployment, the template path specified in the Registration Properties dialog must be defined as follows.

**To configure the policy server user interface for MSR**

1. In the SiteMinder Policy Server User Interface, select the System tab.
2. Click the Registration Schemes object.
3. Select Edit, Create Registration Scheme.
   The Registration Properties dialog opens.
4. Configure a password policy as described in the *Policy Design Guide*.
5. In the Template Path field, specify a path as follows:
   /siteminderagent/selfreg

Proxy Rules for an MSR Request

The SPS supports the Managed Self Registration services through proxyrules.xml for forwarding the request to the Web Agent (6.x) hosting MSR Servlet. The forwarding is based upon the URI of the incoming request. For example, if the URI begins with /siteminderagent/selfreg, the request is forwarded to the Web Agent hosting MSR Servlet; otherwise, the request is forwarded to the backend server.

An example of a proxy rule for forwarding the password services request is following.

```xml
<nete:cond type="uri" criteria="beginswith">
<nete:case value="/siteminderagent/selfreg">
<nete:forward>http://MSR_server.company.com$0</nete:forward>
</nete:case>
<nete:default>
<nete:forward>http://default_backendserver.company.com$0</nete:forward>
</nete:default>
</nete:cond>
```

MSR_server.company.com stands for the server behind the DMZ on which the Web Agent hosting MSR Servlet is installed, and default_backendserver.company.com stands for destination server.
Firewall Considerations

When configuring firewalls for the DMZ that will contain the SPS, the SPS uses ports 8007 and 8009 for internal communication. These ports should be protected from access by entities outside of the DMZ.

*Note:* You can change the ports used by the SPS by altering the appropriate directives in the server.conf file.

Keep Alive and Connection Pooling

The SPS is designed to use a connection pool to provide better performance by spreading out the workload generated from initiating server connections. It is recommended for performance reasons that KEEP ALIVE settings should be turned on for destination servers.

All destination server products have individual methods and attributes for managing keep alive settings. These settings should be reviewed and understood when configuring SPS.

HTTP Header Configuration for Sun Java Web Servers

By default, some web servers, such as Sun Java Web servers limit the number of header variables that can accompany a request. You might have to increase this upper limit to accommodate transactions that contain many custom headers.

The server typically returns 413 Request Entity Too Large error is # of headers is greater than allowable maximum. For more information refer to your destination server’s administration guide.

**To change the maximum number of headers**

1. Locate the magnus.conf file for the back-end Sun Java Web server and open it in a text editor.
2. Add or modify the following entry in magnus.conf:
   
   ```
   MaxRqHeaders 50
   ```
   
   Be sure to set the maximum value at a level above the number of headers created by your SPS transactions.
3. Restart the Sun Java Web server so that the changes will take affect.
HTTP Header for SiteMinder Processing with SPS

The SPS introduces an additional layer in the traditional SiteMinder architecture. This layer forwards or redirects all requests to destination servers in the enterprise. When the SPS 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 SPS proxied the request.

The value of the ProxyAgent parameter in the Agent Configuration object must be set to YES to enable sending the SM_PROXYREQUEST HTTP header to the backend.

Note: This header is only added when a request is made for a protected resource.

Handling Encoded URLs

Web servers can process both encoded and decoded normalized or unescaped URLs. How a Web server handles encoded URLs differs based on the type of server. For security reasons, and to provide consistent behavior, the SPS always decodes or normalizes a URI before processing. This provides a universal representation for a single URL, and protects against attempts to exploit the SPS using encoded strings.
Chapter 12: SSL and the Secure Proxy Server

This section contains the following topics:

Keys and Server Certificates Management (see page 167)
SSL Configuration for FIPS COMPAT and MIGRATE Modes (see page 171)
SSL Configuration for FIPS ONLY Mode (see page 172)
Enable SSL for Virtual Hosts (see page 173)

Keys and Server Certificates Management

The SPS fully supports the Secure Sockets Layer (SSL) protocol. SSL provides secure communication between client and server, enabling mutual authentication (using certificates) and private encrypted messages (using keys).

The SPS uses the OpenSSL cryptography toolkit, which implements the SSL v2/v3 and Transport Layer Security (TLS v1) network protocols and related cryptography standards required by these protocols. The OpenSSL toolkit includes the openssl command line tool for generating keys and certificates. The openssl executable image and supporting libraries are located in the <install dir>\SSL\bin folder or corresponding UNIX directory.

Note: To enable SSL on Solaris, you must have patch 127127-11 installed on the same system as the Secure Proxy Server.

To run the openssl command line tool, connect to the appropriate folder or directory. Open a command line Window or UNIX shell. Use the following syntax as a guideline for entering openssl commands:

openssl command [ command_opts ] [ command_args ]

The openssl tool provides a large number of commands (command in the synopsis above); each one can include numerous options and arguments (command_opts and command_args in the synopsis). You can find complete documentation for openssl at the following URL:

http://www.openssl.org/docs/apps/openssl.html

Important! When you issue the openssl command for any propose be sure to specify a valid path to the openssl configuration file (openssl.conf) using the -config parameter in the command line.
The commands you are most likely to use to perform fundamental SSL tasks are as follows:

- Generating a private key
- Generating a Certificate Signing Request (CSR)
- Generating a certificate by self signing the CSR
- Having a certificate signed by a Certificate Authority (CA)
- Installing a signed certificate
- Decrypting an RSA key
- Encrypting an RSA key
- Changing the password of an RSA key

Before you proceed, review the following important information about private keys and server certificates:

- The server certificate and private key work together. You must use the server certificate with the corresponding private key.
- The server certificate should be digitally signed by a Certificate Authority (CA) or self-signed with your own private key (recommended for sites intended exclusively for internal use).
- The SSLCertificateFile and SSLCertificateKeyFile directives in the SSL.conf file must point to the corresponding certificate and key files.
- If you are using Apache’s virtual host feature, each virtual host you want to secure must have its own private key and server certificate.

### Generate a Private RSA Key

SSL uses keys to encrypt and decrypt messages. Keys generally come in pairs: one public key, and one private key. With OpenSSL, the private key contains the public key information, so you do not generate a public key separately.

Keys use various cryptographic algorithms and key exchange methods. For generating private keys for use with certificates, you most commonly will use the RSA key exchange method with the Date Encryption Standard (DES) cryptographic algorithm in an openssl command (on UNIX in this example) as follows:

```bash
openssl genrsa -des3 -out server.key
```

The key output file will be in encrypted ASCII PEM (from “Privacy Enhanced Mail”) format.
Because the file is encrypted, you will be prompted for a pass-phrase to protect it and decrypt it later if you want. If you do not want your key to be protected, do not use the -des3 argument in the command line.

**Important!** Do not use the -des3 option if you are running on Windows. The Secure Proxy Server does not start if there is a prompt for a pass-phrase.

To view the details of this RSA key, enter the following command:

```
openssl rsa -noout -text -in server.key
```

**RSA Key Decryption**

To remove the encryption from a private key

1. Make a copy of the encrypted key as a backup, for example:
   ```
cup server.key server.key.org
   ```
2. Enter this command:
   ```
   openssl rsa -in server.key.org -out server.key
   ```

Specifying the output file without a preceding encryption option (that is, -des, -des3, or -idea), the file is written out in plain text, and there is no prompt for a pass-phrase.

**Important!** The availability of an unencrypted key on your system makes your system vulnerable to impersonation on the Internet. Make sure that this file has the appropriate permissions, that is, readable only by root on UNIX, or Administrator on Windows.

**RSA Key Encryption**

To encrypt an unencrypted RSA key, enter the following command:

```
openssl dsa -in server.key -des3 -out server.key.new
```

Do not use this command on Windows, because the Web server will not start if there is a prompt for a pass-phrase.

**Modify the Passphrase for an RSA Key**

To change the password on an existing RSA key, enter the following command:

```
openssl rsa -des3 -in server.key -out server.key.new
```

You are prompted for both the old pass-phrase and a new pass-phrase. You then rename your newly created key to the old key name.
Create Certificate Signing Request

Certificates are created for authentication. They associate a public key with the identity of a user or server. Because OpenSSL uses private keys to generate public keys, the first step for creating a certificate is to generate a private key, as described in the previous sections.

The next step is to generate a certificate request, or Certificate Signing Request (CSR), using the private key. You can send the CSR to a Certificate Authority for signing into a certificate, or you can create a self-signed certificate (not recommended, except for testing or other internal use).

To create a CSR with the RSA server private key, enter the following command:

openssl req -config openssl.cnf -new -key server.key -out server.csr

You are prompted for several answers to identify the request.

Note: This command presupposes the existence of an openssl configuration file in the present working directory. The file is located at <install dir>\SSL\bin\openssl.cnf. If you change the name, or move it to another location, you must supply the correct location of openssl.cnf in the command line.

The CSR output file will be in ASCII PEM Privacy Enhanced Mail (PEM) format. You can specify a different format with the -outform option. See the online guide for a list of supported formats.

To view details about the CSR, use the following command:

openssl req -noout -text -in server.csr

Create a Self-Signed Certificate

To create a certificate for testing or other internal purposes, use the following command:

openssl -req -new -x509 -key server.key -out cert_name.crt

To set an expiration time, you can use the -days flag. For example, -days 365 will force the certificate to expire in one year.

Place the output file in the following directory:

sps_home\SSL\certs

You must restart the SPS to enable the certificate.
Obtain Certificate Signed by a CA

To have a certificate signed by a Certificate Authority, go to the CA’s web site and complete the online submission form. You will probably also have to pay a fee. For more information about commercial CAs, you can visit one of these web sites:

- Verisign
  http://digitalid.verisign.com/server/apacheNotice.htm
- Thawte
  http://www.thawte.com/certs/server/request.html

Allow 5-10 working days for the CA to process your request.

Install a Signed Certificate

You install a CA-signed certificate by editing the ssl.conf file. Be sure that the SSLCertificateFile and SSLCertificateKeyFile directives are pointing to the key file and the certificate file you previously created. The csr file is no longer required.

SSL Configuration for FIPS COMPAT and MIGRATE Modes

The procedure for enabling SSL on the SPS varies slightly depending on the FIPS mode. In a new installation with FIPS in COMPAT or MIGRATE mode, configure SSL with the steps listed following. With a new installation or migration to FIPS in ONLY mode, additional steps are required.

To configure SSL in FIPS COMPAT or FIPS MIGRATE mode

1. Generate a Private RSA Key (This is also referred as server key):
   
   openssl genrsa -des3 -out server.key

2. Remove the encryption from a private key:
   
   1. Make a copy of the encrypted key as a backup, for example:
      
      copy server.key server.key.org
   
   3. Enter this command for removing encryption:
      
      openssl rsa -in server.key.org -out server.key

3. Generate a Certificate Signing Request (CSR):
   
   openssl req -config openssl.cnf -new -key server.key -out server.csr

4. Have the certificate signed by a Certificate Authority (CA).

5. Install the signed certificate.
SSL Configuration for FIPS ONLY Mode

For an installation of the SPS in FIPS ONLY mode, the required configuration for SSL support is listed following.

To configure SSL in FIPS ONLY mode

1. Verify that OPENSSL_FIPS environment variable is set to 1 and that the CA_SM_PS_FIPS140 environment variable is set to ONLY.

2. Generate a server key. Specify the size of key as at least 1024 KB. Be sure that the algorithm (des3 in the example following) is FIPS-compliant. For example:
   
   openssl genrsa -des3 -out server.key 1024

3. Generate a Certificate Signing Request (CSR) as shown in this example:
   
   openssl req -config openssl.cnf -new -key server.key -out server.csr

4. Have the certificate signed by a Certificate Authority (CA).

5. Install the signed certificate.

6. Verify that the value of the SSLPassPhraseDialog variable in the httpd-ssl.conf file (located in sps_home\httpd\conf\extra folder) is set to custom.

7. Verify that httpd-ssl.conf file is pointing to correct directives/paths of server key and certs.

8. Enable SSL on the SPS:

   On UNIX:
   
   sps_home/secure-proxy/proxy-engine/sps-ctl startssl

   On Windows:
   
   sps_home\httpd\bin\configssl.bat -enable


The SPS is configured for SSL.

If at a later time you want to run without SSL, enter this command:

   sps_home\httpd\bin\configssl.bat -disable.
7. Verify that the value of the SSLCustomPropertiesFile variable is set to 
   `<sps_home>/Tomcat/properties/spsssl.properties`.

8. Enable SSL on the SPS as follows:
   
   **On UNIX**
   
   1. Enter the following command:
      ```
      sps_home/secure-proxy/proxy-engine/configssl.sh passphrase
      ```
      
      **Note:** The passphrase is the same one provided to the key in Step 2.
      
      This command encrypts the passphrase and stores it in `spsssl.properties` file.
   
   2. Enter the following command:
      ```
      sps_home/secure-proxy/proxy-engine/sps-ctl startssl
      ```
      
      SSL is enabled.

   **On Windows**
   
   1. Enter the following command:
      ```
      sps_home\httpd\bin\configssl.bat –enable passphrase
      ```
      
      **Note:** The passphrase is the one provided to the key in Step 2.
      
      This command encrypts the passphrase and stores it in the `spsssl.properties` file.
   
   2. Restart the Secure Proxy Service.
      
      SSL is enabled.

      **Note:** If at a later time you want to run without SSL, enter the following command:
      ```
      sps_home\httpd\bin\configssl.bat -disable.
      ```

---

**Enable SSL for Virtual Hosts**

The Apache server supports virtual hosts, which are multiple Web hosts that are run from a single Apache binary. Apache virtual hosts can be name-based or IP-based. Name-based virtual hosts can share a single IP address, while IP-based virtual hosts require a different IP address for each virtual host.

Apache virtual hosts using the SSL protocol:

- Must be IP-based due to limitations in the protocol.
- Must have virtual host containers in the Apache configuration file for both secure (HTTPS) and not secure (HTTP) requests.
The following is an example of a secure (HTTPS) virtual host:

```html
<VirtualHost 10.0.0.1:443>
    DocumentRoot ".../htdocs/site1"
    ServerName www.site1.net
    ServerAdmin webmaster@site1.net
    ErrorLog logs/covalent_error_log_site1
    TransferLog logs/...
    SSLEngine on
    SSLCertificateFile /www.site1.net.cert
    SSLCertificateKeyFile /www.site1.net.key
    CustomLog logs/cipher_log_site1 \
            "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x %r\" %b"
</VirtualHost>
```
Session Scheme API

The SPS supports a Java API that allows you to develop custom session schemes. These schemes can be assigned to user agent types for each virtual host configured on the SPS.

Overview of Session Scheme API Processing

The SPS processes a number of methods to establish, maintain, and end a typical user session. One of the steps during session processing is to determine whether a scheme is rewritable. Rewritable schemes provide the ability to modify the URL. The simple URL rewriting session scheme is an example of a rewritable scheme, since part of the processing of a request includes rewriting the requested URL to include a token.

To implement a rewritable session scheme, you must implement the rewritable interface, which is described in Rewritable Session Schemes (see page 180).
The following illustration shows the process flow for the session scheme API methods.

The methods identified in the illustrated are:

1. **isValidRequest**—This method must be implemented in a custom session scheme to determine and verify the conditions that make up a valid request.

2. **getKeyFromRequest**—This method must be implemented to extract a key from a valid request. If no key is present, the createKeyFromRequest method is called.

3. **createKeyFromRequest**—This method must be implemented to trigger the creation of a key for a new session.

4. **onSessionCreate**—On the event of session creation, if the session scheme in use is not rewritable, this method is called. This method may be implemented with any code that should be triggered at the inception of a new session.

5. **onSessionCreateRedirect**—On the event of session creation, if the scheme is rewritable, this method is called. This method may be implemented with any code that should be called at the inception of a new session for a rewritable session scheme.
6. **onSessionUpdate**—A session is updated for each new request made during the session. This method is called during each session update. It may be implemented by adding any code that should be triggered during a session update.

7. **onSessionLogout**—This method is called when a session is terminated. It may be implemented with any code that should be executed when a user session is terminated.

### Session Scheme API Class Files

The SPS session scheme API makes use of the session scheme abstract class contained in `sps_home/Tomcat/server/lib/proxycore.jar`.

### Constructor for Session Scheme API

The constructor for a custom session scheme must consist of the following:

```java
public IPAddrSessionScheme(String name, boolean acceptFlag, Hashtable props) {
    // Always call the parent constructor for proper
    // initialization of the scheme
    super(name, acceptFlag, props);
}
```

The settings are as follows:

#### name

String that is populated by the name in the server.conf file associated with your custom session scheme class.

#### acceptFlag

Boolean value that determines whether or not the custom session scheme accepts SiteMinder’s SMSESSION cookies.

#### props

List of name/value pairs for any other properties required by the session scheme as specified in the server.conf file.
# Session Scheme API Methods

The session scheme API class consists of the following methods:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Method</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>acceptsCookies()</td>
<td>Retrieves the value of the acceptFlag from the accepts_smsession_cookies parameter of the session scheme in the server.conf file and returns a value indicating whether this scheme supports SiteMinder SMSESSION cookies.</td>
</tr>
<tr>
<td>abstract java.lang.String</td>
<td>createKeyFromRequest(HttpServletRequest req)</td>
<td>Executes code to retrieve values needed to create a new session key from the request.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getName()</td>
<td>Retrieves the name of the custom session scheme as defined in the server.conf file.</td>
</tr>
<tr>
<td>abstract Boolean</td>
<td>isValidRequest(HttpServletRequest req)</td>
<td>Evaluates if the request for this session scheme is valid.</td>
</tr>
<tr>
<td>abstract int</td>
<td>onSessionCreate(java.lang.String id, HttpServletRequest req, HttpServletResponse resp)</td>
<td>Hook that is available on the event of session creation.</td>
</tr>
<tr>
<td>abstract void</td>
<td>onSessionLogOut(HttpServletRequest req, HttpServletResponse resp)</td>
<td>Hook that is available at the event of a session termination (logout).</td>
</tr>
<tr>
<td>abstract void</td>
<td>onSessionUpdate(HttpServletRequest req, HttpServletResponse resp)</td>
<td>Hook that is available on the event of session updates. This method is only for internal use.</td>
</tr>
<tr>
<td>static Boolean</td>
<td>usingDefaultSessionScheme(HttpServletRequest req)</td>
<td>Helper method for recognizing that a request is using the default session scheme.</td>
</tr>
</tbody>
</table>
Implement a Custom Session Scheme

Use the following procedure to implement a custom session scheme.

To implement a custom session scheme using the session scheme API

1. Review the sample code for the IP address session scheme in IP Address Session Scheme.
2. Write source code for your session scheme.
3. If you are creating a rewritable session scheme, you must implement the rewritable interface.
4. Ensure that your system CLASSPATH includes the following:
   - proxycore.jar which contains the session scheme API
   - JDK version 1.4 or greater jar files
   - sps_home/Tomcat/server/lib jar files
5. Compile the session scheme.
6. Do one of the following:
   - Create a .jar file that contains your custom session schemes then copy the .jar file to the directory sps_home/Tomcat/server/lib.
   - Add the class files for your custom session scheme to the sps_home/Tomcat/server/classes directory then configure scheme in the SPS server.conf file.
7. Restart the SPS.

Configure Custom Session Scheme in the server.conf File

When you compile the code for a custom session scheme you must configure the session scheme in the SPS server.conf file. To configure the session scheme, add a SessionScheme element to the file. For example:

```xml
<SessionScheme name="custom_scheme">
    <class>com.netegrity.proxy.session.CustomScheme</class>
    <accepts_smsession_cookies>false</accepts_smsession_cookies>
    <property1>value1</property1>
    <property2>value2</property2>
</SessionScheme>
```

In addition, if you have configured user agent types, you can map the session scheme to any appropriate user agents types.

More information:

Session Scheme Mapping for the Default Virtual Host (see page 110)
Configure Rewritable Session Schemes

If a session scheme must have the ability to modify the URL requested by a user, you must implement the rewritable interface. For example, this interface is used by the simple URL rewriting scheme to enable the session scheme to append a token to the end of URL requests.

Implement the Rewritable Interface

When implementing the rewritable interface, the following methods are available:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public string</td>
<td>rewrite(String url, String id, HttpServletRequest req)</td>
<td>Rewrites a requested URL to contain a session identifier.</td>
</tr>
<tr>
<td>public string</td>
<td>onSessionCreateRedirect(String id, String url, HttpServletRequest req, HttpServletResponse resp)</td>
<td>Provides a callback for the event of session creation by redirection. It is typically used in conjunction with forms-based authentication, where the target URL is different from the requested URL. For example, the authentication scheme may modify the URL or add a cookie.</td>
</tr>
</tbody>
</table>

In addition to the rewritable interface, the methods must be implemented in the custom session scheme.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected void</td>
<td>setRequestURI(HttpServletRequest req, String uri)</td>
<td>Allows the scheme to modify the request URI.</td>
</tr>
<tr>
<td>protected void</td>
<td>setRequestPathInfo(HttpServletRequest req, String pathInfo)</td>
<td>Allows the scheme to modify the path information of the request.</td>
</tr>
</tbody>
</table>

The methods described in the following table have been implemented to provide logging for rewritable schemes.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected void</td>
<td>logDebug(String msg)</td>
<td>Logs a debug level message (log level 2)</td>
</tr>
<tr>
<td>protected void</td>
<td>logInfo(String msg)</td>
<td>Logs an information level message (log levels 1-2)</td>
</tr>
<tr>
<td>protected void</td>
<td>logWarn(String msg)</td>
<td>Logs a warning level message (log levels 0-2)</td>
</tr>
</tbody>
</table>
Use an IP Address Session Scheme

The default SPS installation includes an IP address session scheme. This scheme maps a session using the IP address of the client. When a user makes a request, the SPS retrieves the client's IP address from the HTTP headers and uses this to generate the session key for the client's session.

The IP address session scheme was created using the session scheme API. The source code for this scheme can be found in the directory `sps_home\secure-proxy\proxy-engine\examples\sessionschemes`.

Note: In the sample session scheme file, a backslash (\) character indicates that the line should continue, but must be interrupted due to space constraints in this document.

To implement an IP address session scheme

1. Add a `<SessionScheme>` section to your server.conf file like the following:

   ```
   <SessionScheme name="ip_address">
   
   class="com.netegrity.proxy.session.IPAddrSessionScheme"
   
   accepts_smsession_cookies="false"
   
   allowed_proxied_addresses="true"
   
   </SessionScheme>
   ```

   The directives are:

   class
   
   This directive specifies the Java class that handles IP address session schemes. This value should not be modified if you want to use the default IP address session scheme installed with the SPS.

   Default: `com.netegrity.proxy.session.IPAddrSessionScheme`

   accepts_smsession_cookies
   
   Indicates that SiteMinder smsession cookies are not supported by this session scheme. To ensure a cookieless session using the IP address scheme, the value of this directive should not be changed.

   Default: `false`

   allowed_proxied_addresses
   
   Indicates whether or not requests will be validated using the `SessionScheme.isValidRequest` call. Set the value to true to allow the use of proxied addresses. Accept the default, false, to use the isValidRequest method for determining if the VIA HTTP header variable is present. If this variable is present, the SPS determines that the address is proxied and blocks the request.

   Default: `true`
2. Map the session scheme to one or more user agents for a virtual host in the server.conf file.
3. Restart the SPS.

**Session Storage API**

The SPS stores mappings from a session token to a SiteMinder session. This information is accessed using the SessionStorageAPI.

The SessionStorageAPI provides the following capabilities:

**Session creation**
- Allows the creation of a new session.

**Session update or synchronization**
- Allows updates to SiteMinder session information.

**Session retrieval**
- Allows the retrieval of session information when provided with the correct session key.

**Explicit session removal**
- Allows the removal of a session using a specific session key.

**Session expiration**
- Allows the removal of all expired sessions.

**More Information**

Proxy Service Configuration (see page 87)

**Filter API Overview**

Custom filters are filters defined by customer’s needs. SPS uses custom filters to manipulate a request before forwarding the request to a backend server, and also to manipulate the responses sent by the backend server to the user client.

The SPS can process a single custom filter or a group of custom filters for each request. When you create a custom filter group, the SPS processes all the filters that are part of the custom filter group in a chain.
You can look at the source code for a pre-processing filter and a post-processing filter produced with the filter API. These samples may be found in the following directory:

`sps_home/proxy-engine/examples/filters`

**Note:** In the code samples, a backslash (\) character indicates that the line should continue, but must be interrupted due to space constraints in this document.

**More Information**

[Associate Custom Filters to Proxy Rules](#) (see page 183)

### How SPS Processes Custom Filters

The SPS includes an API for inserting pre-processing and post-processing into the proxy stage of a request.

In a standard SPS transaction, the following process occurs:

1. User requests a resource.
2. SPS examines its proxy rules and determines where to direct the request (after successful authentication and authorization).
3. Destination server sends the requested resource to the SPS, which passes the resource to the user.

The Filter API provides a method for developers to insert processing before a request is passed to a destination server, as described in step 2 of the preceding process, or after the response from the destination server is returned to the SPS as described in step 3 of the preceding process, but before the resource is passed to the user.

### Associate Custom Filters to Proxy Rules

When the SPS receives a request or a response, the SPS reads the proxy rules and processes the associated filters. The custom filters or custom group filters that are declared in the server.conf file must be associated with proxy rules. To associate custom filters or custom group filter to proxy rules, open the proxyrules.xml file located in `<install dir>/secure-proxy/proxy-engine/conf`, edit the proxyrules.xml file for the rule that is expected to run the filter.

For example:

```
<nete:forward filter="your filter name or your groupfiltername">http://FQDN$0</nete:forward>
```
Filter API Class File

The SPS Filter API makes use the proxy filter classes contained in `sps_home/Tomcat/server/lib/proxyrt.jar`.

ProxyFilter Interface

The ProxyFilter interface defines the interface implemented by a proxy filter. However, it is recommended that you extend the BaseProxyFilter Abstract Implementation, rather than implementing the ProxyFilter interface.

The ProxyFilter interface consists of the following methods:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Method</th>
<th>Parameters</th>
<th>Throws</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>doFilter(ProxyRequest prequest, ProxyResponse presponse)</td>
<td>Performs the filtering.</td>
<td>ProxyFilterException - thrown if failure processing filtering.</td>
</tr>
<tr>
<td></td>
<td>request - the proxy request data</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>response - the proxy response data</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>request - the proxy request data</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>response - the proxy response data</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProxyFilterConfig</td>
<td>getFilterConfig()</td>
<td>Returns this filter's ProxyFilterConfig object. (ProxyFilterConfig object that initialized this filter).</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>init(ProxyFilterConfig config)</td>
<td>Called when the filter is created to perform any required initialization.</td>
<td>ProxyFilterException - thrown if failure initializing this filter.</td>
</tr>
<tr>
<td></td>
<td>config - a ProxyFilterConfig object containing the filters's configuration and initialization parameters</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
BaseProxyFilter Abstract Implementation

The Filter API includes BaseProxyFilter, an abstract implementation of a proxy filter that can be implemented as a subclass to create ProxyFilters.

**Note:** It is recommended that you extend the BaseProxyFilter Abstract Implementation, rather than implementing the ProxyFilter interface.

A subclass of BaseProxyFilter must override at least one of the following methods:

- doPreFilter
- doPostFilter
- doFilter (not recommended)

The BaseProxyFilter includes filter initialization and separates pre-processing and post-processing hooks for inserting your own filters into SPS transactions, as listed in the following table.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Void</td>
<td>doFilter(ProxyRequest prequest, ProxyResponse presponse) throws ProxyFilterException</td>
</tr>
</tbody>
</table>

This implementation determines the state of the request processing and calls doPreFilter if it's in an inbound state otherwise it calls doPostFilter for outbound state. At the time the filters get called processing can only be in one of these states.

Specified by:
doFilter in interface ProxyFilter

Parameters:
request - the proxy request data
response - the proxy response data

Throws:
ProxyFilterException - thrown if failure processing filtering
### Filter API Overview

**Return Value** | **Method** |
--- | --- |
Void | doPreFilter(ProxyRequest prequest, ProxyResponse presponse) throws ProxyFilterException  
Performs pre-filtering. Override this method to perform filtering tasks before the request is sent to the target server.  
Parameters:  
request - the proxy request data  
response - the proxy response data  
Throws:  
ProxyFilterException - thrown if failure processing filtering |

Void | doPostFilter(ProxyRequest prequest, ProxyResponse presponse) throws ProxyFilterException  
Performs post-filtering. Override this method to perform filtering tasks after the response is received from the target server.  
Parameters:  
request - the proxy request data  
response - the proxy response data  
Throws:  
ProxyFilterException - thrown if failure processing filtering |

ProxyFilterConfig | getFilterConfig()  
Returns this filter's ProxyFilterConfig object.  
Specified by:  
getFilterConfig in interface ProxyFilter  
Returns:  
ProxyFilterConfig the ProxyFilterConfig object that initialized this filter.
### Filter API Overview

#### Chapter 13: SPS APIs

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Void</td>
<td>init(ProxyFilterConfig config) throws ProxyFilterException</td>
</tr>
<tr>
<td></td>
<td>Called when the filter is created to perform any required initialization.</td>
</tr>
</tbody>
</table>

**Note:** When overriding this method, the first statement should call the parent init method "super.init(config).".

Specified by:

- init in interface ProxyFilter

Parameters:

- config - a ProxyFilterConfig object containing the filters's configuration and initialization parameters

Throws:

- ProxyFilterException - thrown if failure initializing this filter.

---

### ProxyFilterConfig Interface

Defines the interface to the configuration data available to a filter. The interface consists of the following methods:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>getFilterName()</td>
</tr>
<tr>
<td></td>
<td>Returns the name of this filter.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getInitParameter(java.lang.String name)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns a String containing the value of the named initialization parameter, or null if the parameter does not exist.</td>
</tr>
<tr>
<td></td>
<td>Parameters:</td>
</tr>
<tr>
<td></td>
<td>name - a String specifying the name of the initialization parameter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.util.Enumeration</th>
<th>getInitParameterNames()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the names of the filter’s initialization parameters as an Enumeration of String objects, or an empty Enumeration if the filter has no initialization parameters.</td>
</tr>
</tbody>
</table>
**ProxyResponse Interface**

Defines the interface that provides access to HTTP response information to be returned to the proxy client. The interface consists of the following methods:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Method</th>
</tr>
</thead>
</table>
| void         | `addHeader(java.lang.String name, java.lang.String value)`<br>Adds a header with the specified name and value. This method allows response headers to have multiple values.  
Parameters:<br>name - a String specifying the header name  
value - a String specifying the header value |
| byte[]       | `getContent()`<br>Returns a byte array of the content of the response to the proxy request. This is the content to be returned to the proxy client. |
| java.lang.String | `getHeader(java.lang.String name)`<br>Returns the value of the specified header as a String. If the header does not exist, this method returns null. The header name is not case sensitive.  
Parameters:<br>name - a String specifying the header name |
| java.utilEnumeration | `getHeaderNames()`<br>Returns an Enumeration of all the header names. If no headers exist, this method returns an empty Enumeration. |
| int          | `getStatusCode()`<br>Returns the HTTP response status code of the response to the proxy request. |
| java.lang.String | `removeHeader(java.lang.String name)`<br>Removes the specified header. Returns the value of the removed header as a String. If the header does not exist, this method returns null. The header name is not case sensitive.  
Parameters:<br>name - a String specifying the header name |
### Return Value

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setContent(byte[] content)</td>
<td>Sets the content of the response to the proxy request. This overwrites the content to be returned to the proxy client.</td>
</tr>
<tr>
<td>void setHeader(java.lang.String name, java.lang.String value)</td>
<td>Sets a header with the specified name and value. If a header with the same name exists it will be overwritten.</td>
</tr>
</tbody>
</table>

### ProxyFilterException Class

The `ProxyFilterException` class defines a general exception that a filter can throw when it encounters difficulty.

<table>
<thead>
<tr>
<th>Constructor Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProxyFilterException()</td>
<td>Constructs a new <code>ProxyFilterException</code>.</td>
</tr>
<tr>
<td>ProxyFilterException(java.lang.String message)</td>
<td>Constructs a new <code>ProxyFilterException</code> with the specified message.</td>
</tr>
<tr>
<td>ProxyFilterException(java.lang.Throwable rootCause)</td>
<td>Constructs a new <code>ProxyFilterException</code> with the specified message and root cause.</td>
</tr>
</tbody>
</table>
# ProxyRequest Interface

Defines the interface that provides access to HTTP request information to be sent by the proxy. The interface consists of the following methods:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>java.lang.String</code></td>
<td><code>getHeader(java.lang.String name)</code></td>
<td>Returns the value of the specified header as a String. If the header does not exist, this method returns null. The header name is not case sensitive. Parameters: name - a String specifying the header name</td>
</tr>
<tr>
<td><code>java.util.Enumeration</code></td>
<td><code>getHeaderNames()</code></td>
<td>Returns an Enumeration of all the header names. If no headers exist, this method returns an empty Enumeration.</td>
</tr>
<tr>
<td><code>javax.servlet.http.HttpServletRequest</code></td>
<td><code>getOriginalRequest()</code></td>
<td>Returns the original HttpServletRequest made to the proxy.</td>
</tr>
<tr>
<td><code>java.lang.String</code></td>
<td><code>getSessionKey()</code></td>
<td>Returns the value of the session key as a String. If the key is not available, this method returns null. The key may be used to rewrite URL's in the content when using cookieless schemes. Note: The SessionScheme is responsible for creating the key and storing it in an attribute named SessionScheme.DEFAULT_SESSION_KEY_NAME</td>
</tr>
<tr>
<td><code>java.lang.String</code></td>
<td><code>getTargetQueryString()</code></td>
<td>Returns the query string the proxy will use with the target URL. The query string may be from the original request or a new one defined through the proxy rules. This method returns null if the URL does not have a query string.</td>
</tr>
<tr>
<td><code>java.lang.String</code></td>
<td><code>getTargetURL()</code></td>
<td>Returns the URL the proxy will use to make the request as defined by the proxy rules. The URL does not include query string parameters.</td>
</tr>
<tr>
<td>Return Value</td>
<td>Method</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>boolean</td>
<td>isInbound()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Returns a boolean value indicating the state of the request processing. If the request has not been forwarded to the target server true is returned. If the request was sent and the response received false is returned.</td>
<td></td>
</tr>
<tr>
<td>boolean</td>
<td>isOutbound()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Returns a boolean value indicating the state of the request processing. If the request has not been forwarded to the target server false is returned. If the request was sent and the response received true is returned.</td>
<td></td>
</tr>
<tr>
<td>java.lang.String</td>
<td>removeHeader(java.lang.String name)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Removes and returns the value of the specified header as a String. If the header does not exist, this method returns null. The header name is not case sensitive. Parameters: name - a String specifying the header name</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>setHeader(java.lang.String name, java.lang.String value)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sets a header with the specified name and value. If a header with the same name exists it will be overwritten. Parameters: name - a String specifying the header name value - a String specifying the header value</td>
<td></td>
</tr>
<tr>
<td>byte[]</td>
<td>getContent()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Returns a byte array of the content of the Request POST data. This is the content which is sent to the backend server.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>setContent(byte[] content)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sets the content of the POST data to the proxy request. This overwrites the content to be sent to the backend server. Parameters: content - the byte array containing the request POST data</td>
<td></td>
</tr>
</tbody>
</table>
Implement a Filter

Filters that use the session key depend on the session scheme to define the key. To make the session key available to a filter, an attribute keyed by SessionScheme.DEFAULT_SESSION_KEY_NAME must be set to hold the value of the key when it is created by the createKeyFromRequest(..) callback and retrieved on subsequent requests by the getKeyFromRequest(..) callback of the Session Scheme API.

Out-of-the-box session schemes that generate the session key are:

- Mini-cookies
- Simple URL rewriting

To implement a filter using the Filter API

1. Review the sample code for filters in Filter Examples.
2. Write source code for your filter.
3. Ensure that your system CLASSPATH includes the following:
   - proxyrt.jar which contains the filter API
   - JDK version 1.4 or greater jar files
   - sps_home/Tomcat/server/lib jar files
4. Compile the filter.
5. Do one of the following:
   - Create a .jar file that contains your filter and copy the file to the directory sps_home/Tomcat/server/lib directory.
   - Add the class files for your filter to the sps_home/Tomcat/server/classes directory, in a subdirectory that corresponds to the package name.
6. Configure the SPS server.conf file.
7. Edit the proxyrules.xml file for the rule that is expected to implement the filter. For example:
   ```xml
   <nete:forward filter="your filter name">http://FQDN$0</nete:forward>
   ```
8. Restart the SPS.

Filter API Example

The SPS installation includes sample source files for a preprocessing filter and a post-processing filter. Both of these samples use the BaseProxyFilter Abstract Implementation. For a complete description of the example filters, see Filter Examples.
Using a Filter to Rewrite Absolute Links in a Requested Page

One of the most common uses of the Filter API is to support the rewriting of absolute links in pages requested by a user through the SPS. For absolute links to be handled properly by the SPS, you must use the Filter API to perform the appropriate substitution for any absolute links contained in resources returned to the user based on a SPS request.
Chapter 14: Troubleshooting

This section contains the following topics:

Unable to Start Apache on UNIX systems (see page 195)
DNS Caching in the SPS (see page 196)
No Root Permissions (see page 197)
Cannot Start the SPS Server (see page 197)
Cannot Access the SPS with a Browser (see page 198)
Unknown Server Name (see page 198)
Issues Configuring Virtual Hosts (see page 199)
Command not found Error Received (see page 199)
SPS Not Forwarding Requests (see page 199)
SPS and SharePoint (see page 200)

Unable to Start Apache on UNIX systems

Symptom:
When running the SPS on a UNIX system, the Apache server fails to start. In the Apache log file, the following error message appears:

Invalid argument: setgid: unable to set group id to ...

Solution:
This error occurs when the group for the Run-As-User on UNIX systems does not correspond to the group specified in the Apache configuration file (httpd.conf). If you see this error, edit the Group directive in the Apache httpd.conf file.

To edit the Group directive
1. Remove the comment sign (#) before the Group directive
2. Specify the group to which the Run-As-User belongs.
3. Run the SPS startup command again (sps-ctl start or startssl).
DNS Caching in the SPS

Symptom:
I do not want the SPS to cache the DNS name look-up settings of the server.

Solution:
The SPS is configured by default to always cache the DNS settings of the server. To change this default behavior, adjust the networkaddress.ttl setting in the java.security file.

To modify how long the SPS caches the DNS settings
1. Navigate to the directory sps_home\secure-proxy\JDK\1.5.0\jre\lib\security.
2. Open the java.security file.
3. Set the networkaddress.cache.ttl parameter to a positive integer. For example, networkaddress.cache.ttl=2

networkaddress.ttl

Specifies the duration, in seconds, for which the SPS caches the successful DNS name look-ups. The value for this directive should be any positive integer. A negative value indicates that the SPS will always cache the DNS settings.

Default: -1
No Root Permissions

Symptom:
Configuring the SPS without root permissions.

Solution:
Use the following information to troubleshoot:

- You can still install SPS; however, the automatic process cannot complete all of the installation steps. The Installation Program displays warnings to help you determine which files you need to edit by hand.

- Non-root users are generally not allowed to bind to ports 80 and 443. Therefore, the default ports for HTTP and HTTPS traffic are set to 8080 and 8443 for non-root installations. This enables you to test your server without editing the httpd.conf file. Test with:
  - http://yourserver.com:8080
  - https://yourserver.com:8443

  Note: Non-root installations are not recommended for SSL-enabled servers. A non-root installation is less secure because it allows an additional person with root permissions access to your keys and certificates.

Cannot Start the SPS Server

Symptom:
SPS server fails to start.

Solution:
Use the following information if you cannot start your server:

- Verify that the ServerName directive in `sps_home/secure-proxy/httpd/conf/httpd.conf` corresponds to the name of your server.

- Verify that the server is not already running by executing one of the following:
  - `ps -ax|grep http` on BSD compatible systems
  - `ps -elf|grep http` on System V release 4 compatible systems

  If this results in a list of processes, stop the running server before starting your new server.

- Check the log files in the directory `sps_home/secure-proxy/httpd/logs`
Verify that the SSLCertificateFile and the SSLCertificateKeyFile directives in the httpd.conf file point to your certificate and key files. The file is in the directory `sps_home/secure-proxy/httpd/conf`

- Determine whether you are using non-IP-based virtual hosts. SSL requires IP-based virtual hosts.
- Verify that no other server is running on the default port for the SPS. The default port is specified in the httpd.conf file.
- If you using SSL, be sure you have generated a key and certificate before starting the server, otherwise you will get an error.

### Cannot Access the SPS with a Browser

**Symptom:**
Difficulty accessing the SPS using a browser.

**Solution:**
To access the SPS using a browser:

- Verify that DNS is aware of your servername with the command `nslookup servername` or try to ‘ping’ your server with the `ping servername` command.
- Run the server without SSL and access your web site to verify whether the problem is with the key or certificate files. To start the server without SSL, execute `./sps-ctl start` in the directory `sps_home\secure-proxy\proxy engine directory`.
- Try to make a telnet connection to ports 80 and 443 of your Web server (or the non-default ports you specified). If you installed as a non-root user, try to connect to ports 8080 and 8443.

### Unknown Server Name

**Symptom:**
Determining the server name.

**Solution:**
Do any one of the following to determine your server name:

- Use the `hostname` command.
- Enter `echo $DISPLAY` at a command prompt. If the result is `servername:0.0`, your servername is the portion that precedes the colon.
- Consult the guide for your operating system.
Issues Configuring Virtual Hosts

Symptom:
Difficulty configuring virtual hosts.

Solution:
Refer to the information about configuring virtual hosts at:

http://httpd.apache.org/docs-2.0/vhosts/

Command not found Error Received

Symptom:
Command not found error,

Solution:
- Use the ls command to verify that the command exists in the directory you are in and that it is spelled correctly.
- Add ./ before the command, for example,./setup.
- If you are using Solaris and encounter problems running the SPS, verify that you have installed all recommended operating system patches for your version of Solaris. For more information, see the release notes for the SPS.

SPS Not Forwarding Requests

Symptom:
404 File Not Found browser error.

Solution:
If you receive a 404 File Not Found browser error and there is no action in the Web Agent log for the SPS Web Agent, verify the name an IP address of the virtual host in the server.conf file.
SPS and SharePoint

Symptom:
When accessing a SharePoint page through the SPS, the SPS always displays the Alternate Access Mapping Connection parameter, no matter how the mode (Forward or Redirect) is set in the proxyrule.xml file.

Solution:
Follow these steps for a solution to this problem:

1. On the SharePoint server, go to Central Administration, Operation, Alternate Access Mapping. Notice that the Alternate Access Mapping includes a Default Zone Internal URL and a Public URL.

2. Add one Internal URL with the Public URL set as http://<SPS Host>:port and Default Zone.

3. Add one more Internal URL Public URL set as http://<SharePoint Host>:port and Default Zone.

4. Edit the entry for the Intranet zone created in step 3 and specify the Public URL as http://<SPS Host>:port

5. In the SPS proxyrule.xml file, the backend is an internal URL with a public URL pointing to the SPS host. For example:

```xml
<!--Proxy Rules-->  
<nete:proxyrules xmlns:nete="http://ww.ca.com/">
    <nete:forward>http://SharePointServer with public URL as SPS host:port$0</nete:forward>
</nete:proxyrules>
```
Index

A
acceptFlag • 177
acceptsCookies • 178
access control • 11, 16, 18, 22, 145
access devices • 20
addHeader • 188
Agent • 153, 163
Agents • 153
Agents, SiteMinder Application Server • 17
Apache • 71, 167
API
    filter • 192
    API, session scheme • 16
architecture
    proxy server • 11
    reverse proxy server • 12
ATTLIST • 125
B
Basic authentication schemes • 154
beginswith • 124, 125
C
cacertfilename • 82
cacertpath • 82
cache
    session • 85
    session store • 85
caching • 12, 17, 145
cases • 119
Certificate Authority • 167
    trusted • 82
ciphers • 82
clean_up_frequency • 85
clusters • 145, 146
comparison type attribute • 122
compatibility • 156
condition • 19
conditions • 119
configuration
    simple proxy server • 11
connection pool • 87, 91
constructor
    session scheme API • 177
containers, virtual host • 173
cookie • 16, 20
cookie-less • 20
    session schemes • 146, 158
    sessions • 16, 20
    single sign-on • 16, 18, 20
createKeyFromRequest • 175, 178
D
destination • 119
destination servers • 113
destinations • 119
device ID • 16, 20
device ID session scheme • 101
DMZ • 113, 155, 160, 162, 165
Documentation Type Definition
    proxy rules • 120
doFilter • 184, 185
doPostFilter • 185
doPreFilter • 185
DTD • 120
dynamic header value • 137
E
elements • 119
enableredirectrewrite • 113
endswith • 125
entitlements • 18, 19
equals • 125
error
    413 Request Entity Too Large • 165
    messages, logging • 78
extranet • 22, 156
F
fatal errors, logging • 78
federation gateway • 45, 63
file descriptors • 87
filter • 87, 128
filter API • 192
firewall • 11, 19
forward • 12, 87, 92, 124, 131
forwards • 20
G

gateway • 11, 12, 16, 17
getContent • 188
getFilterConfig • 184, 185
getFilterName • 187
getHeader • 188, 190
getHeaderNames • 188, 190
getInitParameter • 187
getInitParameterNames • 187
getKeyFromRequest • 175, 178
getName • 178
getOriginalRequest • 190
getSessionKey • 190
getStatusCode • 188
getTargetQueryString • 190
getTargetURL • 190

H

header • 119, 125, 140
HTTP • 19
SiteMinder • 19
headers • 147
host • 119, 125
hostname • 19, 117, 142
HTML forms authentication schemes • 154
HTTP • 19, 74, 87, 147
and HTTPS • 16
header • 19, 125, 140
listener • 74
HTTP header • 119
header • 147
http_connection_pool_incremental_factor • 87
http_connection_pool_max_attempts • 87
http_connection_pool_max_size • 87
http_connection_pool_min_size • 87
http_connection_pool_timeout • 87
http_connection_pool_wait_timeout • 87
HTTPS • 19, 87, 147
httpsd.conf • 71, 146

I

init • 184, 185
in-memory session store • 146
intranet • 22, 156
IP address • 20
session scheme • 97
IP-based virtual hosts, requirement for SSL • 173

isinbound • 190
isOutbound • 190
isValidRequest • 175, 178

J

JDK 1.3 • 179

K

keep-alives • 91

L

LDAP search expression • 160
load balancing devices • 145
local content • 17
logappend • 78
logDebug • 180
logfile • 78
logfilename • 78
logging • 34, 74
logInfo • 180
loglevel • 78
logWarn • 180

M

Managed Self-Registration • 162
mapping
session scheme • 12
max_size • 85
mini-cookies • 16, 20
session scheme • 97
MSR • 162, 163

N

name-based virtual hosts • 173
nete

case • 122, 125, 127
cond • 121, 122, 125, 127
default • 127
description • 122
forward • 121, 122, 128, 136, 137
local • 121, 122
proxyrules • 121
debug • 121
redirect • 121, 122, 129, 136, 137
result • 130, 131, 135, 136, 137
rule • 131, 135
xpr • 130, 132, 135
xprcond • 122, 130, 132, 135
xpr-default • 130, 132
nspsapi.jar • 179

O
OnRejectRedirect • 158
onSessionCreate • 175, 178
onSessionCreateRedirect • 175, 178
onSessionLogout • 175, 178
onSessionUpdate • 175, 178

P
password policies • 160
password services • 160
plug-in • 17
policies • 17, 153, 156
policy domains • 153
Policy Server • 19, 22, 153, 163, 164
post-processing • 192
pre-processing • 192
private key • 167
props • 177
protection level • 154, 158
protocol.multiple • 87
protocals • 87
Proxy Engine • 11, 12, 16, 71, 74, 132
proxy rules • 12, 16, 18, 19, 20, 87, 138
DTD • 119, 120
hostname • 142
regular expressions • 143
URI • 141
proxy server
reverse • 11, 12
traditional • 11
Proxy Trust Mode • 156
ProxyFilterException • 189
proxyrules.dtd • 138
proxytimeout • 155
proxytrust • 155

Q
query • 125
query string • 119, 132

R
realms • 153, 156
redirect • 12, 20, 87, 92, 124, 131, 136, 158
redirect URL • 113
redirectrewritablehostnames • 113
regular expression • 19, 130, 132, 135, 143
removeHeader • 188, 190
resource cache • 12
responses • 138, 153, 158
SiteMinder • 138
reverse proxy server • 11, 12
traditional • 12
rules • 153, 156
proxy • 12, 16
rules_file • 138

S
Secure Sockets Layer • 82
server certificate
See also certificate • 167
server configuration • 71
server connectors • 74
server.conf • 34, 71, 74, 78, 82, 85, 87, 120, 138, 146, 148, 165
services configuration • 71
servlet • 163
session scheme • 16, 146
device ID • 101
IP address • 97
mappings • 34
mini-cookies • 97
SSL ID • 96
session scheme API • 16
acceptFlag • 177
acceptsCookies • 178
createKeyFromRequest • 175, 178
getKeyFromRequest • 175, 178
getName • 178
isValidRequest • 175, 178
name • 177
onSessionCreate • 175
onSessionCreateRedirect • 175, 178
onSessionLogout • 175, 178
onSessionUpdate • 175, 178
props • 177
rewritable interface
LogDebug • 180
LogInfo • 180
LogWarn • 180
setRequestPathInfo • 180
setRequestURI • 180
usingDefaultSessionScheme • 178
session schemes • 11, 12, 20, 71
  cookie-less • 16
session store • 16, 17, 20, 146
session store cache • 85
sessions scheme API
  onSessionCreate • 178
sessions storage • 20
SessionStore • 85
setContent • 188
setHeader • 188, 190
setRequestPathInfo • 180
setRequestURI • 180
single sign-on • 16, 20, 22, 145, 146
  cookie-less • 16
SiteMinder • 18, 22, 153, 156, 158, 162
  cookie • 146
  header • 19
  objects • 153
  responses • 138
Web Agents • 16
SiteMinder Application Server Agents • 17
SiteMinder responses • 138
SM_PROXYREQUEST • 166
SM_REWRITE_URL • 158
sockets • 87
SSL • 82, 146
  versions • 82
SSL acceleration • 12
SSL ID • 16, 20
SSL ID session scheme • 96
sslparams • 82
SSLv2 • 82
SSLv3 • 82
Template Path • 164
timeout • 87, 91
TLSv1 • 82
token • 16, 20
trusted
  Certificate Authority • 82
URI • 119, 124, 125, 132, 135, 141
  matching • 122
  substring • 19, 124
URL • 119, 124
URL rewriting • 16, 20
user agent • 71
user directories • 153
usingDefaultSessionScheme • 178
V
value attribute • 125
virtual host • 20
virtual host configuration • 113
virtual hosts • 34, 71, 167
W
Web Agent • 19, 160, 162
  settings • 34
Web Agents • 17, 22, 153, 155
Web Agents, SiteMinder • 16
Web servers • 153
WebAgent.conf • 155
worker.ajp13.port • 74
X
XML • 119
XML configuration file • 120