CA™ SOA Security Manager

SOA Security Gateway Configuration Guide

r12.1

CA Product References

This document references the following CA products:

- CA SOA Security Manager
- CA SiteMinder® Web Access Manager

Contact CA

Contact Technical Support

For online technical assistance and a complete list of locations, primary service hours, and telephone numbers, contact Technical Support at http://ca.com/support.

Provide Feedback

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

If you would like to provide feedback about CA product documentation, please complete our short customer survey, which is also available on the CA Support website.

Contents

2. General Configuration

Getting Started

Overview

The purpose of this guide is to offer the reader a step-by-step guide to getting started with SOA Security Gateway. From installing and setting up the SOA Security Gateway to configuring security policies, this document should act as a first port of call for SOA Security Gateway administrators and users.

SOA Security Gateway Overview

Before installing SOA Security Gateway, the reader is advised to take a look at the brief overview of the product. Furthermore, it is important to check the System Requirements and Release Notes to ensure that your target machine and platform are supported. This information and more can be found by following these links:

- SOA Security Gateway Overview
- System Requirements
- Release Notes

Install SOA Security Gateway

Having read the SOA Security Gateway overview and reviewing the System Requirements, you are now in a position to install the product. SOA Security Gateway is available for Windows, Linux, and Solaris. Please refer to the SOA Security Gateway install guide for instructions on how to install the product on your chosen platform.

Startup Instructions

The correct startup sequence for the components of SOA Security Gateway is as follows:

1. Start SOA Security Gateway
2. Start SOA Security Gateway Management Console

The startup scripts for these components can be found in the `/bin` directory of your product installation.

Global Configuration

Global configuration options, such as the "Web Services Repository", "Certificates", and "Users", are available as top-level nodes in the tree view of the SOA Security Gateway Management Console. In general, these options are configured globally so that they can be referenced within individual message filters. For example, connection details for an LDAP directory can be configured globally and then referenced from different instances of message filters. This saves the administrator the effort of re-keying the connection details every time a connection to the LDAP directory is required.

For more information on global configuration options, please select the relevant link from the list below:

- Global Configuration
- Web Services Repository
- Configuring Processes
- Configuring HTTP Services
- Messaging Systems
- Global Cache
- Directory Scanner

Message Filters

There are many categories of message filters available with SOA Security Gateway, including authentication, authorization, content filtering, signing, conversion, and many more. The help pages for these categories of filters can be found under the **Message Filters** section of the main Index page of the documentation.

A useful index of all message filters can be found at the following page:

- Message Filter Reference

More Help

Context sensitive help is available from all screens on the SOA Security Gateway Management Console. Simply click the **Help** button on any screen to display the relevant help page for that screen.

The most up-to-date version of the product documentation is always available on the

CA extranet.

If you require further information or assistance, please contact the Support Team.

Support

It is important to include as much information as possible when sending support emails to the CA Support Team. This will help us diagnose and solve the problem in a more efficient manner. The following information should be included with any support query:

- Name and version of the product, e.g. SOA Security Gateway.
- Details of any patches that were applied to the product, if any.
- Platform on which the product is running.
- A clear (step-by-step) description of the problem or query.
- If you have encountered an error, the error message should be included in the email. It is also useful to include any relevant trace files from the `/trace` directory of your product installation, preferably with the trace level set to DEBUG.

CA SOA Security Gateway Overview

Overview

The CA SOA Security Gateway is a dedicated network device for offloading processor-intensive tasks from applications running in general purpose application servers. The CA SOA Security Gateway performs application networking by routing traffic based on content, based on sender, and by performing XML content screening. XML data is converted on the fly between formats.

The gateway can be used to offload the heavy-lifting of XML from application servers and onto the network. This frees up resources on application servers and allows applications to run faster. The patented high performance core VXA (CA XML Acceleration) engine, coupled with hardware acceleration ensures wirespeed network performance. The CA SOA Security Gateway also includes a comprehensive suite of threat prevention message filters.

The following sections describe some of the high-level functionality available in CA SOA Security Gateway.

Ease of Deployment

As an appliance, the gateway is pre-hardened and requires no installation. It can be managed in a "headless" manner. A Web-based interface is also provided. All policies can be imported and exported as XML files, which minimizes the time taken to replicate policies across multiple devices, or to move from a staging system to production environment.

The gateway includes many features that speed up deployment. For example, certificates and private keys, necessary for many XML security functions, may be issued onboard. The device has a "Deny by Default" defence posture, in order to detect and block any unauthorized deployments of Web Services. Policies can be re-applied across multiple application endpoints using simple drop-down menus.

Powerful Rules Engine

An intuitive policy management console enables administrators to add security and management policies to the gateway device. Policies across multiple gateway devices may be managed together using CA's SOA Security Gateway Management Console enterprise policy management product. This allows enterprise policy management to be brought under centralized control, rather than being managed on a device-by-device basis.

Processing Offload

Offload the heavy-lifting of XML from application servers and onto the network. XML operations such as XML Schema Validation and XSLT are notoriously slow. The gateway uses patented, wirespeed, hardware acceleration to speed up these tasks. This frees up resources on application servers and allows applications to run faster.

Application-level Netorking

The gateway routes data based on sender identity, content, and content type. This allows XML messages to be sent to the appropriate application. It also allows for "service virtualization" to be performed, whereby Web Services are exposed to clients with "virtualized" addresses, which mask their actual addresses for security and application-delivery reasons.

XML Data Enrichment

Automatically populate content in XML documents from sources such as databases. By putting this functionality onto XML Networking infrastructure, the information is automatically populated into the XML messages before they reach the consuming Web Services. This simplifies and accelerates applications in ESBs or Application Servers.

Identity Management

CA SOA Security Gateway can use an existing Identity Management infrastructure to perform authentication and authorization of XML traffic. Integration is provided with LDAP, Microsoft Active Directory, CA SiteMinder, Entrust GetAccess, IBM Tivoli Access Manager, Oracle COREId and RSA Access Manager and other IM products. The gateway also interoperates with leading XML products and platforms, including Microsoft .NET, BEA WebLogic, IBM WebSphere, and SAP NetWeaver.

Identity Mediation

Through support for a wide range of security standards, CA SOA Security Gateway allows for identity mediation between different identity schemes. For example, the gateway can authenticate external Web Services clients using passwords but then issue SAML tokens that are used for identity propagation to application servers.

Service Virtualization

CA SOA Security Gateway serves as an important control point for XML traffic on the network. By shielding end point Web Services from direct access, the gateway allows for the virtualization of these services and clients access the SOA Security Gateway as if it was the Web Service itself. This allows different "views" of Web Services to be presented to different clients.

Traffic Throttling

The CA SOA Security Gateway protects Web Services from unanticipated traffic spikes by smoothing out the traffic. It also limits clients to agreed Web Service consumption levels in accordance with service usage agreements. This allows CA's customers to charge their clients for different levels of Web Services usage.

Audit Capability/Compliance

Satisfy audit requirements by allowing Web Services transactions to be archived in a tamper-proof store for subsequent audit. CA also facilitates privacy compliance support by allowing sensitive information, such as customer names, to be encrypted or stripped out of XML traffic.

Pluggable Pipeline

The gateway's internal message-handling pipeline is extensible, allowing extra access control and content-filtering rules to be added with ease. Customers do not have to wait for a full product release before receiving updates of support for emerging standards and for additional adapters.

VX Deployment Platform

Integrated into the gateway is CA's patented core VXA (XML security acceleration) engine. This processing engine accelerates the essential XML security primitives.

SOA Security Gateway Release Notes

Overview

The following release notes describe installation, running, and other known issues for CA SOA Security Gateway 5.1. The latest version of this document is always available on the CA extranet.

What's New

The following table outlines the list of features that are new in the SOA Security Gateway 5.1:

Theme	***New Features***
Policy and Configuration Management	• Role-based management of policies • Configuration Rollback • Configuration Versioning • Configuration Access Control • Configuration life cycling support • Policy Comparison and Merging tool support • Policy creations wizards to help the design of policies • Web Services Repository for easy configuration of policies using WSDL files.
Transport Independence	• HTTP inbound/outbound • JMS inbound/outbound • Directory/File inbound/outbound • Protocol mediation, e.g. HTTP inbound / JMS outbound, directory inbound / HTTP outbound, JMS inbound / HTTP

Theme	***New Features***
	outbound
Testing	• Improved SOAPbox • WSI Compliance Testing • Performance Testing Utilities
Monitoring and Reporting	• Enhanced Real Time monitoring capabilities • Live dashboards • On demand policy execution profiling
Integration	• Improved Tivoli Access Manager support • Increased UDDI V2 support • Systinet Integration
General	• Response Caching • Packet sniffing on a network interface • Servlet deployment

Installation

Do not install into a directory that contains the following characters: `';?@=&$-_+!*'()'`

HTTP Port Number

By default, the SOA Security Gateway is configured to run on ports 8080 and 8090. If

the machine(s) on which these processes are installed already has services running on these ports, alternative ports must be configured for the SOA Security Gateway. This can be done by right-clicking on the process HTTP interface in the SOA Security Gateway Management Console and selecting the Edit option

If the SOA Security Gateway is being installed on a machine that already has services running on port 8080, for example, Tomcat, then the SOA Security GatewaySOA Security Gateway Management Console

Running on Linux

To run on Linux Red Hat, you need to change several libraries. The package extension is dependent on the architecture of the Linux machine.

- For SPARC architecture, the naming convention is `"package name".sparc.rpm`.
- For Intel architecture, the naming convention is `"package name".i386.rpm`.

Red Hat/Suse/etc

The following libraries are required:

- `bzip2-1.0.1-3`
- `compat-bzip2-0.9.5d-5cl`
- `compat-glibc-6.2-2.1.3.2`
- `db1-1.85-4`
- `db2-2.4.14-4 RPM`
- `db3-3.1.14-6`
- `libstdc++.so.6 from gcc 4.1.2 or higher`
- `libc.so.6, from glibc 2.3 or higher`
- `libpthread.so.0, from glibc 2.2 or higher`
- `libgcc_s.so.1, from gcc 4.1.2 or higher`
- `libdl.so.2 from glibc 2.1 or higher`
- `rpm-4.0-4`

- `rpm-build-4.0-4`
- `rpm-devel-4.0-4`

It is important to note that using a version of glibc lower than 2.3 may cause the JVM to crash.

Running on Solaris 8

If you are starting any of the components of an SOA Security Gateway installation on Solaris 8 and the component fails to start with the following error:

```
You must install a Solaris patch to run the native threads version of
the Java runtime.  The green threads version will work without this
patch.  Please check the native threads release notes for more
information.  If you are embedding the VM in a native application,
please make sure that the native application is linked with
libthread.so (-lthread).
```

Then set the `LD_PRELOAD` to point to the `libthread.so` library, for example to start the SOA Security Gateway Management Console:

```
env LD_PRELOAD=libthread.so ./soagatewaymanagementconsole
```

Documentation

The documentation cannot be viewed from the CD when the CD-ROM driver adheres strictly to the ISO 9600 standard. This is the international standard for CD-ROM file format, and does not allow filenames longer than 8 characters, or filenames containing capital letters. As a result several of the document links are broken. This problem should only manifest itself when reading the documentation from the CD on Solaris machines. The work around here is to copy the documentation on to the local drive when you wish to view the documentation.

Increasing Performance

You can increase performance by increasing the heap size that the JVM is using. CA recommend the use of a large Java heap size to run the SOA Security Gateway. Enabling a larger heap size allows the SOA Security Gateway to process large volumes of traffic more efficiently. In general, the maximum Java heap size should be set to no more than half the RAM size on the machine running the SOA Security Gateway. This can be done for the JVM using the following command line arguments:

Java Argument	***Effect***
-Xms64M	Sets the minimum heap size to 64 Mb
-Xmx256M	Sets the maximum heap size to 256 Mb

To increase the heap size for the SOA Security Gateway, these arguments should be added to the `jxm.xml` file located in the `INSTALL_DIR/system/conf` directory, where `INSTALL_DIR` refers to the root of your SOA Security Gateway installation. To set the minimum heap size to 256 Mb add the following line:

```
<VMArg name="-Xms256M" />
```

To set the maximum heap size to 512 Mb, add the following line:

```
<VMArg name="-Xms512M" />
```

Simply add these command line arguments as before to the `soagateway.xml` file.

Acknowledgements

This product includes software developed by the Apache Software Foundation [http://www.apache.org/]

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/) [http://www.openssl.org/].

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)

This product includes software developed by James Cooper.

Latest Information

For the latest information on the SOA Security Gateway, take a look on the CA extranet.

To obtain your username and password to log in to this secure area of the Web site please contact the Sales Team

SOA Security Gateway System Requirements

Components in the SOA Security Gateway

- **SOA Security Gateway:**

  The SOA Security Gateway applies security policies to incoming and outgoing XML messages. The security policies applied can facilitate authentication, authorization, or content filtering.

- **SOA Security Gateway Management Console:**

  An intuitive management tool that allows an administrator to easily configure security policies (authentication, authorization, content filtering, etc) to secure Web Services.

- **Monitoring Console:**

  For remote real-time monitoring of activity within the SOA Security Gateway. This allows an administrator to detect malicious activity in real time, and to take precautionary actions if they feel a service is under attack.

Requirements

This section describes the system requirements for the SOA Security Gateway

Requirement	***Supported platforms***
Operating System	• Microsoft Windows 2000 Server • Microsoft Windows 2000 Advanced Server • Microsoft Windows XP Professional • The SOA Security Gateway is functional under Windows 2000 Professional or

Requirement	***Supported platforms***

	Windows XP but this environment is not recommended by CA due to performance impairment.
	• Sun Solaris 8, 9, and 10.
	• Most recent Linux distributions, it has been tested on the following distributions:
	  • Redhat Linux 7.2 Kernel 2.4.7-10
	  • Redhat Linux 8.0 Kernel 2.4.18-14
	  • Redhat Linux 9.09 Kernel 2.4.20-8
	  • Red Hat Enterprise Linux WS release 3 (Taroon Update 4) Kernel 2.4.21-27.EL
	  • Debian 3.0 Kernel 2.4.20
	  • Fedora Core, release 1 Kernel version: 2.4.22-1.2115.nptl
	  • SuSe 9.0 Kernel version 2.4.21
	  • SuSe 9.1.Kernel version 2.6.4

Processor	• On Windows: Pentium II, III, IV, or Pentium compatibles • On Solaris: Sun Sparc 32-bit • On Linux: Pentium II, III, IV, or Pentium compatibles.

Specific requirements:

Component	***Requirement***
SOA Security Gateway Management Console	As above, plus Solaris/Linux requires X-Windows environment.

Component	***Requirement***
Monitoring Console	As above, plus Solaris/Linux requires X-Windows environment.
Server	Min 256 Mbyte RAM, max RAM dependent on expected server load. 450MB disk space

Global Configuration

Overview

For convenience, the SOA Security Gateway Management Console contains a tree view of global configuration options. For example, there are libraries of users, X.509 certificates, and schemas that can be added globally and then referenced in filters and policies. This avoids the need to re-configure details over and over again each time the schema or certificate (for example) is to be used.

The following global libraries are available on the tree view, each of which will be discussed briefly in the sections below:

- Policy Store
- Web Services Repository
- Processes
- Policies
- Certificates
- Users
- Alerts
- External Connections
- Schema Cache
- Blacklist
- Whitelist
- Caches

Policy Store

The top level node in the tree view of the SOA Security Gateway Management Console represents the underlying Policy Store where all configuration information is stored. It is labeled in the tree view according to the URL at which it resides. For example, typically the HTTP URL of the Policy Store is displayed since this is the default deployment option.

Take a look at the Policy Store Configuration Options help page for more information on how to configure the various features that are available at this level.

Web Services Repository

The easiest way to secure a Web Service with CA SOA Security Gateway is to import the WSDL (Web Services Description Language) file for the service using the SOA Security Gateway Management Console. In doing this, a policy will be created for the service consisting of policy resolvers (i.e. **SOAPAction**, **SOAP Operation**, and **Relative Path**) and the relevant connection filters.

The WSDL file is also added to the Web Services Repository, making sure to update the URL of the Web Service to point at the machine on which the SOA Security Gateway is running as opposed to that on which the Web Service is running. Consumers of the Web Service can then query the SOA Security Gateway for the WSDL file for the Web Service. The consumer then knows to route messages to the SOA Security Gateway instead of attempting to route directly to the Web Service, which most likely will not be available on a public IP address.

The Web Services Repository offers administrators a very simple way of securing a Web Service with minimal impact on consumers of that service. Because of this, the Web Services Repository should be used as the primary method of setting up policies within the SOA Security Gateway Management Console. For more information on using the Repository to setup policies, take a look at the Web Services Repository tutorial.

Processes

A Process represents a single running instance of a CA SOA Security Gateway. It allows you to configure at least 2 interfaces - one for public traffic and a second for listening for and serving configuration data. The configuration interface should rarely need to be updated, however, it is likely that you will want to add several HTTP interfaces. For example, you may want to add a HTTP interface and also an SSL-enabled HTTPS interface.

Furthermore, it is possible to add JMS listeners, packet sniffers, and directory scanners to a Process once it has been added. This allows the SOA Security Gateway to listen for JMS messages, inspect packets at the network level for logging and monitoring purposes, and scan messages that have been dumped to the file system.

Because the SOA Security Gateway can read messages from HTTP, JMS, or a directory, gives it the ability to perform protocol translation. For example, it is possible to read a message from a JMS queue and then route it on over HTTP to a Web Service. Similarly, the SOA Security Gateway can read XML messages that have been FTP-ed to a directory on the file system and send them to a JMS messaging system or route them over HTTP to a back-end system.

For more information on configuring processes and the various transport interfaces,

please refer to the Processes tutorial.

Policies

A policy is made up of a sequence of modular, reusable message filters, each of which processes the message in a particular way. There are many categories of filters available, including authentication, authorization, content filtering, routing, and many more.

So, for example, a typical policy might contain an authentication filter, followed by several content-based filters (e.g. Schema Validation, Threatening Content, Message Size, XML Complexity, etc), and provided all configured filters run successfully, the message will be routed on to the configured destination.

A policy can be thought of as a network of message filters. A message can traverse different paths through the network depending on what filters succeed or fail. This allows us to configure policies that, for example, route messages that pass one Schema Validation filter to one back-end system, and route messages that pass a *different* Schema Validation filter to a *different* system.

To help manage your policies, *Policy Containers* can be used. A Policy Container is typically used to group together a number of similar policies (e.g. all authentication policies) or to act as an umbrella around several policies that relate to a particular policy (e.g. all policies for the "getQuote" Web Service.

A number of useful policies that ship with the SOA Security Gateway can be found under the "Policy Library" Policy Container. This container is pre-populated with policies to return various types of faults to the client and policies to block certain types of threatening content, amongst others. It is, of course, possible to add your own policies to this container, as it is to create your own Policy Containers as necessary to suit your own requirements.

The "Configuration" Policy Container is used to store the authentication and (non-editable) content-based filtering that is applied to configuration messages. It should only be changed under strict supervision from the CA support team.

Certificates

CA SOA Security Gateway must be able to trust X.509 certificates in order to establish SSL connections with external servers, validate XML Signatures, encrypt XML segments for certain recipients, and for other such cryptographic operations. Similarly, in order to carry out certain other cryptographic operations, such as message signing and decrypting data, a private key is required.

The **Trusted Certificate Store** contains all the certificates that are considered to be trusted by the SOA Security Gateway. Certificates can either be imported into or created by the Certificate Store. It is also possible to assign a private key to the public key stored in a certificate, either by importing the private key or by generating one using

the provided interface.

For more information on importing and creating certificates, please refer to the Trusted Certificate Store help page.

User Store

Users are mainly used for authentication purposes within CA SOA Security Gateway. In this context, the **User Store** acts as a repository for user information against which users can be authenticated. It is also possible to store user attributes for each user, which can then be used when generating SAML attribute assertions on behalf of the user, for example.

Furthermore, a public and private key pair (and X.509 certificate) can be associated with each user for use in SSL mutual authentication, signing XML messages, signing log messages, signing OCSP and XKMS requests, and XML Encryption, amongst other uses.

The Users help page contains more details on how to create users, create key pairs, import certificates, and assign privileges to them.

System Alerts

The SOA Security Gateway can send system alerts to various error reporting systems in the case of a policy error, for example, when a request is blocked by a policy. Alerts can be sent to an Windows Event Log, UNIX syslog, OPSEC firewall, SNMP NMS, or email recipient.

Please refer to the System Alerts help page for more information on how to configure the SOA Security Gateway to send these alerts.

External Connections

CA SOA Security Gateway can leverage your existing identity management infrastructure, thus avoiding the need to maintain separate "silos" of user information. For example, if you already have a database full of user credentials, the SOA Security Gateway can authenticate requests against this database, rather than using its own internal user store. Similarly, the SOA Security Gateway can authorize users, lookup user attributes, and validate certificates against 3rd party identity management servers.

Each such connection to an external system can be added as a global *External Connection* in the SOA Security Gateway Management Console so that it can be re-used across all filters and policies. So, for example, if you create a circuit that authenticates users against an LDAP directory and then validates an XML signature by retrieving a public key from the same LDAP directory, then it makes sense to create a global External Connection for that LDAP directory. The LDAP Connection can then be merely *selected* in both the authentication and XML signature verification filters, rather than having to be re-configured in both filters.

The following default types of global connections can be configured using the External Connections interface. Other bespoke *Connectors* are available on request.

- Authentication Repository Profiles
- Database Connections
- LDAP Connections
- OCSP Connections
- XKMS Connections

External Connections can also be used in cases where you want to configure a group of related URLs. This is most useful in cases where you want to round-robin between a number of related URLs to ensure high availability. When the SOA Security Gateway is configured to use a *URL Connection Set* (as opposed to just a single URL), it will round-robin between the URLs in the set.

For more information on configuring External Connections and Connection Sets, please refer to the External Connections tutorial.

Schema Cache

The global **Schema Cache** contains all the XML Schemas that the SOA Security Gateway can use to validate incoming requests against. The **Schema Validation** filter validates the format of an incoming message against a schema from the cache. This ensures that only messages of the correct format are processed by the target system.

Refer to the Schema Cache tutorial for instructions on how to import XML Schemas into the cache. Once you have imported your schemas, you can take a look at the Schema Validation tutorial for instructions on how to validate XML messages against the schemas in the cache.

Blacklist and Whitelist

The **Whitelist** is a global library of regular expressions that can be used across several different filters. For example, the **Validate HTTP Headers**, **Validate Query String**, and **Threatening Content** filters all use regular expressions from the **Whitelist** to ensure that various parts of the request contain expected content.

The **Whitelist** is pre-populated with regular expressions that can be used to identify common data formats, such as alphanumeric characters, dates, email addresses, IP addresses, and so on. So, for example, if a particular HTTP header is expected to contain an email address, the "Email Address" expression from the library can be run against

the HTTP header to make sure that it does, in fact, contain an email address as expected. This is just yet another way that CA SOA Security Gateway can ensure that only the correct data reaches the Web Service.

While the **Whitelist** contains regular expressions to identify "good" data, the **Blacklist** contains regular expressions that can be used to identify common attack signatures. For example, there are expressions to scan for SQL injection attacks, buffer overflow attacks, ASCII control characters, DTD entity expansion attacks, and many more.

It is possible to run various parts of the request message against the regular expressions contained in the **Blacklist** library. For example, the HTTP headers, request query string, and message (MIME) parts can be scanned for SQL injection attacks by selecting the SQL-type expressions from the **Blacklist**.

More information on running regular expressions can be found in the help pages for the individual filters, such as:

- Validate HTTP Headers
- Validate Query String
- Validate Message Attributes
- Threatening Content

Caches

It is possible to configure the SOA Security Gateway to cache responses from a back-end Web Service. So, for example, if the SOA Security Gateway receives 2 successive identical requests it can (if configured) take the response for this request from the cache instead of routing the request on to the Web Service and asking it to generate the response again.

As a result, excess traffic is diverted from the Web Service making it more responsive to requests for other services, the SOA Security Gateway is saved the processing effort of routing identical requests unnecessarily to the Web Service, and the client benefits from the far shorter response time.

Local caches can be configured for each running instance of the SOA Security Gateway. If you have deployed multiple SOA Security Gateways throughout your network, it is possible to configure a distributed cache where cache events on one cache are replicated across all others. So, for example, if a response message is cached at one instance of the SOA Security Gateway, it will be added to all other caches.

For more information on how the SOA Security Gateway can be configured to use local and distributed caches, please refer to the Global Caches help page.

Policy Store Configuration Options

Overview

The top level node in the tree view of the SOA Security Gateway Management Console represents the underlying Policy Store where all configuration information is stored. It is labeled in the tree view according to the URL at which it resides. For example, typically the HTTP URL of the Policy Store is displayed since this is the default deployment option.

The following configuration options are available by right-clicking on the Policy Store tree node and selecting the appropriate menu option.

Edit

By clicking this menu option, the **Set Configuration Name** dialog is displayed. You can enter a friendly name here for the Policy Store, which will then appear in the tree view instead of the URL of the Policy Store.

Refresh Server

Whenever you make changes to a filter or policy using the SOA Security Gateway Management Console you must refresh the server in order for those changes to take affect. The **Refresh Server** menu option can be used to achieve this. If the server is in the middle of processing a number of messages at the time when the server refresh command is issued, these messages will all be processed using the older policy. New messages will be queued until this batch of messages are completely processed. Once the new policy data has been stored and loaded by the server (i.e. the server has fully refreshed) the queued messages will be processed using the new policy.

Disconnect

This option can be used to force the SOA Security Gateway Management Console to disconnect from the current Policy Store. This can be used, for example, in cases where you have multiple Policy Stores and want to switch between configuring these stores.

Default Settings

The **Default Settings** entered here will be applied to all instances of the SOA Security Gateway that are using this particular Policy Store. So, for example, it is possible to change the trace level, timeouts, cache sizes, and other such global information. For more information on each of these settings, please refer to the General Settings help page.

It is important to note that these settings can be overwritten by configuring settings at the Process level. This option is available by right-clicking on the Process in the tree view of the SOA Security Gateway Management Console and selecting the **Settings -> Custom** menu options.

Default Logging Settings

This option allows you to configure the default logging behavior of the SOA Security Gateway. The SOA Security Gateway can be configured to log to a database, file, UNIX syslog, the system console, or the CA Logging Console. For more information on logging to these destinations, take a look at the Logging Configuration help page.

MIME/DIME Settings

The SOA Security Gateway can filter MIME messages based on the content types (or MIME types) of the individual parts of the message. The **MIME/DIME Settings** dialog lists the default MIME types that the SOA Security Gateway can filter on. These types are then used by the **Content Types** filter to determine which MIME types to block or allow through to the back end Web Service. More information on configuring these types can be found in the MIME/DIME Settings help page.

Namespace Settings

The **Namespace Settings** dialog can be used to determine the versions of SOAP, WSSE (Web Services Security), and WSU (Web Services Utility) that the SOA Security Gateway supports. Please refer to the Namespace Settings help page.

Change Passphrase

By default, data is stored unencrypted in the Policy Store. It is, however, possible to encrypt certain sensitive information, such as passwords and private keys, using a passphrase. For information on how to do this, take a look at the help page on Setting the Encryption Passphrase.

Web Service Repository

Overview

The **Web Services Repository** is used to store information about Web Services whose definitions have been imported via the SOA Security Gateway Management Console. The WSDL files that contain these Web Services definitions are stored together with their related XML Schemas. Clients of the Web Service can then query the repository for the WSDL file, which they can then use to build and send messages to the Web Service via the SOA Security Gateway. We will explain how exactly this works later in this tutorial.

When importing WSDL files into the repository, it is possible to auto-generate a policy involving policy resolvers, routing filters, and a schema validation filter. The policy can be imported from the file system, a URL, or from a UDDI registry and can be simultaneously checked for WS-I (Web Services Integration) compliance.

The next sections will describe how to import a Web Service definition into the repository and will show exactly what is created at each step.

Importing the WSDL File

The **Web Services Repository** is available as a top-level configuration option in the tree view in the SOA Security Gateway Management Console. WSDL files are imported into "Web Service Groups", which provide a convenient way to keep groups of related Web Service definitions together. A WSDL file can be imported into the default group by right-clicking on the "Web Services" tree option and selecting the **Import WSDL** menu option.

Alternatively, a new Web Services group can be added by right-clicking on either the default group (i.e. "Web Services") or the "Web Services Repository" tree item itself and selecting the **Add a new Web Services group** option. Once the new group has been added, it is possible to right-click on it in the tree view and select the **Import WSDL** option.

The first screen on the **Import WSDL** wizard dialog allows you to choose the location of the WSDL from the file system, a URL, or from a UDDI registry. Select the appropriate option depending on the location of the WSDL that you want to import. If you want to retrieve a WSDL file from a UDDI directory, take a look at the help page on Retrieving WSDL Files from a UDDI Directory.

When importing the WSDL file it is also possible to check it for compliance with the WS-I Basic Profile. The Basic Profile consists of a set of assertions or guidelines on how to ensure maximum interoperability between different implementations of Web Services. For example, there are recommendations on what style of SOAP to use (e.g. "document/liter-

al"), how schema information should and should not be included in WSDL files, and how message parts should be defined to avoid ambiguity for consumers of WSDL files.

The SOA Security Gateway Management Console can test imported WSDL files for conformance against the recommendations laid out in the Profile. A report is generated showing exactly what recommendations have passed and what ones have failed. While a WSDL file that does not conform to the Profile can still be imported, there is no certainty that consumers of the Web Service will be able to use it without encountering problems.

Important Note:

In order to run the WS-I compliance test, the WS-I Test Assertions Document (TAD) file must be installed on the machine on which the SOA Security Gateway Management Console is running. This file can be obtained from www.ws-i.org [http://www.ws-i.org]. The *full path* to the location of this file (e.g. **c:/wsi/SSBP10_BP11_TAD.xml** must be specified in the global preferences for the SOA Security Gateway Management Console. To configure this setting, select the **Settings -> Preferences** option from the main menu bar of the SOA Security Gateway Management Console. In the **Settings** dialog, select the "WS-I TAD File" setting and then enter the location of the TAD file in the field provided.

To run the WS-I compliance test on an imported WSDL, click the **Run WS-I Compliance Test** button after selecting the WSDL file using the radio buttons on this screen. The results of the compliance test are displayed in the **WS-I Compliance Results** dialog.

The overall result of the compliance test is given in the "Summary Result" column next to the top-level node, which reflects the name of the service. The individual results of the WS-I compliance tests are grouped together by type beneath the top-level node. So, for example, all the test assertions that relate to the <wsdl:portType> elements in the WSDL are listed under the "portType" node, while all assertions relating to the <wsdl:operation> elements are listed under the "operation" node in the results tree.

Specific details about each assertion can be viewed in the **Assertion Details** section by clicking on the ID (e.g. "BP2103") of the assertion in the results tree. The details correspond to the information given for the assertion in the TAD file.

WSDL Operations

The **WSDL Operations** screen of the wizard displays all operations defined in the WSDL file. The "Relative Path", "Binding", and "Namespace" of each operation is also displayed.

Use the checkboxes to select the operations that you want to create policy resolvers for. The SOA Security Gateway Management Console will use the Web Service location,

SOAP operation, and SOAP Action specified in the WSDL to create **Relative Path**, **SOAP Operation**, and **SOAP Action** policy resolving filters.

We will see in the next section how it is possible to remove definitions for unchecked operations from the WSDL file that is stored in the **Web Services Repository**.

WSDL Import Settings

The **Remove unselected operations from the WSDL** checkbox configures whether or not the SOA Security Gateway Management Console will remove unchecked operations from the WSDL file that is stored in the repository. As a result, the removed operations will not be exposed to clients that pull down the WSDL file for the Web Service.

It is important to note that when a request is made to the SOA Security Gateway for a WSDL file that has been imported into the **Web Services Repository**, it will change the address of the Web Service specified in the **location** attribute of the <soap:address> element to point to the machine on which the SOA Security Gateway is running, rather than the machine that is actually hosting the Web Service. This means that when a client pulls down the WSDL file for the Web Service, they will route messages through the SOA Security Gateway instead of attempting to send messages directly to the Web Service, which typically won't be available on a public IP address. Take a look at the Publishing the WSDL section at the bottom of this page for a detailed example of how this works.

Deploy Policy

The final step in the wizard is to select where to deploy the newly created policy. The **Deploy Policy** screen gives a list of all available Services and their corresponding Relative Paths. Select a Relative Path under which the policy (or policies) will be deployed. All requests arriving on the selected path will now be dispatched to the newly created policy.

Click the **Finish** button to complete the configuration. A summary of the created policies will be displayed in a summary dialog containing links. Use the links to view the created policies.

What was created?

Assuming that all the default options were selected in the **WSDL Import** wizard, the following list summarizes what was created by the import wizard:

1. Web Service Group

A new Web Service Group will be created for each imported WSDL. The Group will contain the WSDL file itself, together with any imported resources, such as other WSDL

files or schema files. By clicking on the new Web Service Group in the tree view of the SOA Security Gateway Management Console, the list of imported WSDL files and XML Schemas are listed in the Service summary panels. The Schemas are listed by namespace.

Take a look at the imported WSDL file by clicking on the location of the WSDL file underneath the Web Service Group in the tree view. If you have selected to remove unchecked operations from the WSDL, these operations will have been removed from the stored WSDL.

2. Policy Container

A new container for the 3 new policies will be created under the "Policies" node in the tree view. The new container will be named after the service.

3. Top-level Service Policy

The top-level "Service Policy" is the first of 3 policies created by the wizard. It contains the logic to retrieve the WSDL file for the service if requested by the client. Clients can simply specify the name "WSDL" on the request query string in order to retrieve the WSDL file, for example, **http://localhost:8080/services/getQuote?WSDL"**. If the request is not a request for the WSDL file, it is considered a request for the actual Web Service itself. In this case the request will first be passed to a resolving policy shortcut and then to the routing policy.

4. Resolver Policy

The "Resolver Policy" consists of 3 filters: **Relative Path**, **SOAP Action**, and **SOAP Operation**. This information is taken from the operation's definition in the WSDL file. If the operation is a *document-style* operation, a **Schema Validation** filter will also be created in this policy. The validation filter will point to the schema that was imported from the WSDL file into the **Web Services Repository**. Note that **Schema Validation** filters cannot be generated for RPC-style operations.

5. Routing Policy

The "Routing Policy" consists of a **Static Router** that contains the address of the Web Service as specified in the original WSDL file and a **Connection** filter to actually make the connection to the Web Service.

6. Mapping to Relative Path

Finally, the Relative Path selected on the last screen of the import wizard will be mapped to the top-level "Service Policy". To view this mapping, right-click on the selected Relative Path and select the **Show Policy** menu option.

Publishing the WSDL

Once the WSDL has been imported into the **Web Services Repository**, it can be retrieved by clients. In effect, by importing the WSDL into the repository, we are *publishing* the WSDL. In this way, consumers of the services defined in the WSDL can learn how to communicate with those services by retrieving the WSDL for those services. However, to do this, the location of the service location must be altered to reflect the fact that the SOA Security Gateway now sits between the client and the defined service.

For example, let's assume that the WSDL file states that a particular service resides at http://www.example.com/services/myService, e.g.

```
<service name="myService">
  <port binding="SoapBinding" name="mySample">
    <wsdl:address location="http://www.example.com/services/myService"/>
  </port>
</service>
```

Once deployed behind the SOA Security Gateway, this URL is no longer accessible to consumers of the service. Because of this, clients must send SOAP messages through the SOA Security Gateway in order to access the service. In other words, they must now address the machine hosting the SOA Security Gateway instead of that directly hosting the service.

Once the WSDL file has been published to the repository, clients can retrieve it. However, when returning the WSDL to the client, the SOA Security Gateway will dynamically change the value of the "location" attribute in the "service" element in the WSDL file to point to the machine on which the SOA Security Gateway resides. The details regarding the physical location of the Web Service are preserved in the **Connection** filters, which are responsible for routing messages on to the service.

Assuming that the SOA Security Gateway is running on port 8080 on a machine called "SERVICES", the location specified in the exported WSDL file will be changed to the following:

```
<service name="myService">
  <port binding="SoapBinding" name="mySample">
    <wsdl:address location="http://SERVICES:8080/services/myService"/>
  </port>
</service>
```

When the modified WSDL file is distributed to the client, it will now route messages to the machine hosting the SOA Security Gateway instead of attempting to directly access the Web Service.

In order for the client to get hold of this modified WSDL file, the SOA Security Gateway Management Console provides a WSDL retrieval facility, whereby clients can query the **Web Services Repository** for the WSDL file for a particular Web Service. To do this the client must pass the name "WSDL" on the query string to the Relative Path that is mapped to the policy for this Web Service.

For example, if the policy is deployed under `http://SERVICES:8080/services/getQuote`, the client could retrieve the WSDL for this Web Service by sending a request to `http://SERVICES:8080/services/getQuote?WSDL`. Once the client has a copy of the updated WSDL file, it will know how to create correctly formatted messages for the service and perhaps more importantly, will know to route messages to the SOA Security Gateway rather than to the Web Service directly.

Configuring Processes

Overview

The purpose of this tutorial is to show how to configure the CA SOA Security Gateway Process. The Process represents a running instance of the SOA Security Gateway. The following functionality can be configured at the Process level:

1. **Import WSDL:**

   By importing a WSDL file at the Process level, a Policy comprising policy resolver filters is created. A Relative Path is then added to the "Public HTTP Interfaces" Services Group and then mapped to the new policy.

2. **Add HTTP Interfaces:**

   This option allows you to add a container for HTTP-related services, including HTTP and HTTPS Interfaces, Directory Scanners, Static Content Providers, Servlet Applications, and Packet Sniffers.

3. **Messaging System:**

   The SOA Security Gateway can read JMS messages from a JMS queue or topic, run them through a policy, and then route onwards to a Web Service or JMS queue or topic.

4. **Directory Scanner:**

   The Directory Scanner is used to read XML files from a specified directory and dispatch them to a selected policy.

5. **POP Client:**

   The POP Client can poll a POP mail server and read messages from it. Messages can then be passed into a policy for processing.

6. **Remote Host:**

   Remote Host settings are used to "tweak" the way in which the SOA Security Gateway routes to another host machine.

7. **Reporting Metrics:**

   The SOA Security Gateway can store certain statistics, called Message Metrics, about each message it processes in a database, which can be used by Reporter to produce HTML-based reports and charts.

8. **Crypto Acceleration:**

   The SOA Security Gateway can leverage OpenSSL's Engine API to offload complex cryptographic operations (e.g. RSA and DSA) to a hardware-based crytographic accelerator and also as an extra layer of security when storing private keys on a Hardware Security Module (HSM).

9. **Settings:**

   This option allows you to configure various global properties for the Process.

10. **Logging:**

    A Process can be configured to log messages to a database, the file system, or UNIX syslog. A Log Viewer for examining log entries is also available.

The remainder of this tutorial will discuss each of these configuration steps in turn.

Import WSDL

The **Import WSDL** option on the Process performs the following operations:

1. Creates a "resolving" policy based on information contained within the WSDL file consisting of **Relative Path**, **SOAP Operation**, and **SOAP Action** resolvers. In cases where complete Schema information is present in the WSDL file, a **Schema Validation** filter can also be generated.
2. Adds a Relative Path beneath the "Public HTTP Interfaces" container. The path will correspond to the path used in the **location** attribute of the **soap:address** element. So, for example, if the address of the Web Service is given as **http://www.ca.com/services/getQuote**, the Relative Path will be set to **/services/getQuote**.
3. The newly added Relative Path will be mapped to the "resolver" policy created in the first step above.

By automatically configuring the policy, the path, and then mapping between the two,

the administrator is saved the effort of manually configuring each item. However, it is most likely that the policy will be modified to include more specific checks on ingress XML messages, such as content validation, authentication, authorization, and routing.

HTTP Services

HTTP Services act as a container for all HTTP-related interfaces to the SOA Security Gateway's core messaging pipeline. HTTP and HTTPS interfaces can be configured to accept plain HTTP and SSL messages respectively. A Relative Path interface is available in order to map requests received on a particular URI (or path) to a specific policy. The Static Content Provider interface can retrieve static files from a specified directory, while the Servlet Application allows you to deploy servlets beneath the service. And finally, the Packet Sniffer interface can read packets directly of the network interface, assemble them into HTTP messages, and dispatch them to a particular policy. The HTTP Services help page contains information on how to configure the available HTTP Interfaces.

Messaging System

The SOA Security Gateway can consume a JMS queue or topic as a means of passing XML messages to its core message processing pipeline. Once the message has entered the pipeline it can be validated against all authentication, authorization, and content-based message filters. Having passed all configured message filters, it can be routed to a destination Web Service over HTTP or it can be dropped back on to a JMS queue or topic using the **Messaging System** Connection filter. For more information on configuring the SOA Security Gateway to read messages from a JMS queue or topic, take a look at the Messaging System help page.

Directory Scanner

The **Directory Scanner** allows you to search a local directory for XML files, which can then be fed into a security policy for validation. Typically, XML files are FTP-ed or saved to the file system by another application. The SOA Security Gateway can then pick these files up, run the full array of authentication, authorization, and content-based filters on the messages, and then route them over HTTP or JMS to a back-end system. For more information on the **Directory Scanner** please refer to the Directory Scanner help page.

POP Client

The **POP Client** allows you to poll a POP mail server in order to read email messages from it and pass them into a policy for processing. For more information on the **POP Client** please refer to the POP Client help page.

Remote Hosts

In cases where a destination server may not fully support HTTP 1.1, **Remote Host** settings can be configured for the server to "tweak" the way in which the SOA Security Gateway sends messages to it. Similarly, if the server requires an exceptionally long timeout, this can be configured in the **Remote Host** settings. For more information on how to configure **Remote Hosts**, take a look at the Remote Hosts help page.

Process Settings

Per-process settings are configured by right-clicking on the Process in the tree view of the SOA Security Gateway Management Console and selecting the **Settings** option. For more information on configuring Process settings, take a look at the General Settings help page.

Process Logging

It is possible to configure a Process to log messages to a database, GUI Console, log files, or UNIX syslog. Take a look at the Logging Configuration help page for more information on how to do this.

Reporting Metrics

The SOA Security Gateway can store useful statistics about the messages that it processes in a database. It is then possible for the Reporter monitoring tool to poll this database and produce charts and graphs detailing how the SOA Security Gateway is performing. For more information on configuring reporting metrics, take a look at the Reporting Metrics help page.

Cryptographic Acceleration

The SOA Security Gateway uses OpenSSL to perform cryptographic operations, such as encryption and decryption, signature generation and validation, and SSL tunneling. OpenSSL exposes an *Engine API*, which makes it possible to plug in alternative implementations of some or all of the cryptographic operations implemented by OpenSSL. OpenSSL will, if configured appropriately, call the engine's implementation of these operations instead of its own. For more information on configuring the SOA Security Gateway to use an OpenSSL engine, please refer to the Cryptographic Acceleration help page.

Import and Export Policies

Configuration data for Processes can be imported into and exported from the underlying Policy Store. This is useful in cases where, for example, you have configured Processes in a testing environment and now want to move them to a live production environment. By exporting the Processes from the test environment and then importing them to the production one, the Processes can be effectively migrated.

To import Processes into the Policy Store, right-click on the "Processes" node in the tree view of the SOA Security Gateway Management Console and select the **Import Processes** menu option. Browse to the location of the XML file that contains details for previously exported Processes. The Processes will then be imported.

Similarly Processes can be exported by right-clicking on the "Processes" node and clicking the **Export Processes** menu option. The Processes will be exported to an XML file, which can then be imported into a different Policy Store.

References

OpenSSL Engine API [http://www.openssl.org/docs/crypto/engine.html]

Configuring HTTP Services

Overview

CA SOA Security Gateway uses *HTTP Services* to service traffic from various sources. *Active Services* can receive HTTP traffic from a socket (i.e. a TCP port), from a HTTP request URI (i.e. a relative path), or by serving files from the file system, while *Passive Services* read HTTP traffic directly off the network interface card and are completely transparent to clients.

Because of the way in which they are deployed, Active Services are typically used for policy enforcement, while Passive Services are mainly used for Web Services management and monitoring. The following list summarizes the list of available Active and Passive Services:

HTTP Interfaces:

HTTP Interfaces are used in the SOA Security Gateway to define the ports and IP addresses on which a Process should listen for incoming requests. *HTTPS Interfaces* can also be added, allowing you to select the SSL certificate to use to authenticate to clients and also what certificates are considered trusted for establishing SSL connections with clients. Both HTTP and HTTPS Interfaces are types of Active Service because they open up a TCP port and actively listen for requests on it.

Relative Path:

While HTTP and HTTPS Interfaces open sockets and listen for traffic, *Relative Paths* can be configured so that when a request is received on that path, the Process can map it to a specific policy. Relative Paths are Active Services since they are "listening" for requests on a certain path.

Static Content Provider:

A *Static Content Provider* is another type of Service that can be used to serve static HTTP content from a particular directory. In this case, the Process is effectively acting as a Web Server. Similarly to the Relative Path Service, this Service is processing requests that are received on a particular path and can, therefore, be thought of as a type of Active Service.

Servlet Applications:

The Process can act as a servlet container allowing you to drop servlets into the HTTP Services configuration. This functionality should only be used by developers with very

specific requirements and under strict advice from from the CA support team.

Packet Sniffer

Finally, a *Packet Sniffer* can be added to intercept network packets from the client, assemble these packets into complete HTTP messages, and then log these messages to an audit trail. Since the Packet Sniffer operates at the network layer (as opposed to a HTTP-based traffic monitor, which operates at the application layer), the packets are intercepted transparently to the client. Because of this, the Packet Sniffer is a *Passive Service* and is typically used for SOA management and monitoring.

The rest of this guide describes how to configure the various Services described above: HTTP Interfaces, Relative Paths, Static Content Providers, Servlet Applications, and Packet Sniffers.

Service Groups

A *Service Group* is used as a container around one or more Services. Usually a Service Group will be configured for a particular type of Service. So, for example, it would be typical to have a "HTTP Services Group" that contains the configured HTTP Interfaces, another "Packet Sniffing Group" to manage Packet Sniffers, and maybe a "Directory Scaning Group" to contain the Directory Scanners. While organizing Services by type eases the task of managing Services, the SOA Security Gateway is flexible enough to allow administrators to organize Services according to whatever scheme best suits their requirements.

We will now look at a situation where you might want to use different Service Groups. We will first consider a "HTTP Service Group" that handles HTTP traffic and then show how a second "SSL Service Group" can be used to process SSL traffic on a separate channel.

HTTP Services Groups should consist of at least one HTTP Interface together with at least one Relative Path. The HTTP Interface determines what TCP port the Process listens on, while the Relative Path can be used to map a request received on a particular path (i.e. request URI) to a specific policy.

It is possible to add several HTTP Interfaces to the group, in which case requests received on any one of the opened ports will be processed in the same manner. So, for example, it is possible for requests to **http://[HOST]:8080/test** and **http://[HOST]:8081/test** to be both processed by the same policy, i.e. the one mapped to the "/test" Relative Path.

Similarly, it is possible to add multiple Relative Paths to our HTTP Services Group, where each Path is bound to a specific Policy. So, for example, if a request is made to **http://[HOST]:8080/a** it will be processed by "Policy A"; whereas if a request is

made to **http://[HOST]:8080/b** it will be handled by "Policy B". As a side-effect of this configuration, requests made to the other Interface will also be processed by the same policy - meaning that a request made to **http://[HOST]:8081/b** will also be handled by "Policy B".

Effectively, what this means is that the Relative Paths configured under the HTTP Services Group will be bound to all HTTP Interfaces configured for that group. In other words, if we have 2 Interfaces listening on ports 8080 and 8081, for example, then requests to **http://[HOST]:8080/a** and **http://[HOST]:8081/a** are handled identically by the Process.

But what happens if we want to make a distinction between receiving requests on the 2 different ports? For example, say we want requests to **http://[HOST]:443/a** to be processed by an "SSL Validation" policy, while requests for **http://[HOST]:8080/a** to be handled by a standard "Schema Validation" policy. How can we achieve this?

The addition of a new Services Group can resolve this issue. The new "SSL HTTP Services Group" will open a single HTTPS Interface that listens on port 443 and will be configured with a Relative Path of "/a" to handle requests on this path. The configuration is summarized in the following table:

Services Group	***HTTP Port***	***Relative Path***
HTTP Services Group	8080	/a
SSL HTTP Services Group	443	/a

With this configuration, when we receive a request on **http://[HOST]:8080/a** it will be handled by the "Schema Validation Policy". But when we get a request to the SSL port on **http://[HOST]:443/a** it will be processed by the "SSL Validation Policy".

By using Service Groups in this manner, it is possible to configure the Process to dispatch requests received on the same path (e.g. "/a") to different policies depending on the port on which the request was accepted.

By default, the SOA Security Gateway ships with 2 pre-configured Service Groups - "Management Services" and "Default Services". However, new Service Groups can be added to allow for situations like the one described in the above example where we want to dispatch requests to different policies based on the port on which the requests were received. Similarly, it makes sense to add new Service Groups for managing different types of Services, for example, a "Directory Scanning" group or a "Packet Sniffing" group.

Please see the Note on Management Services section below for more information on the default "Management Services" group.

To add a Service Group, right-click on the Process and select the **Add Service Group**

menu option. Simply enter a name for the group in the **Service Group** dialog.

Once a Service Group has been created, individual HTTP Interfaces, Relative Paths, Static Content Providers, and Packet Sniffers can be added to the group. The remaining descriptions describe how to configure these features.

HTTP Interfaces

A HTTP Interface defines the address and port that the Process listens on. There are two types of Interface: the HTTP Interface and the HTTPS Interface. The HTTP interface handles standard non-authenticated HTTP requests, while the HTTPS interface can accept mutually authenticated connections.

To create a HTTP Interface, right-click on the name of the Service Group (i.e. "Default Services") in the tree view of the SOA Security Gateway Management Console and select **Add Interface** from the menu options. To create a HTTP Interface select the **HTTP** option. To create a HTTPS Interface, select the **HTTPS** menu option.

The following fields must be configured on the **HTTP Interface** or **HTTPS Interface** dialogs:

Port:

The port number that the Process will listen on for incoming HTTP requests.

Address:

This is the IP address or host of the network interface on which the Process will listen. For example, it is possible to configure the Process to listen on port 80 on the external IP address of a machine, while having a Web server running on the same port but on the internal IP address of the same machine. By entering, '*', the Process will listen on all interfaces.

Protocol Version:

Configure the Internet Protocol version that this Interface will use. It is possible to enable support for IPv4, IPv6, or both of these protocol versions.

Backlog:

The number of connections that the Process will allow the operating system to accept before it can process them. There is obviously a trade off here: the larger the back log, the larger the memory usage; the smaller the backlog, the greater the potential for dropped connections.

Idle Timeout:

The SOA Security Gateway supports the use of HTTP 1.1 persistent connections. The **Idle Timeout** is the time that the Process will wait after sending a response over a persistent connection before it closes the connection.

Typically, a client will tell the Process that it wants to use a persistent connection. The Process acknowledges this instruction and decides that it will keep the connection open for a certain amount of time after sending the response to the client. If the client does not reuse the connection by sending up another request within the **Idle Timeout** period, the Process will close the connection.

Active Timeout:

When the Process receives a large HTTP request, it reads the request off the network as it becomes available. If the time between reading successive blocks of data exceeds the **Active Timeout**, the Process will close the connection.

The idea here is to guard against a client closing the connection while in the middle of sending data. Imagine the client's network connection is pulled out of the machine while in the middle of sending data to the Process. When the Process has read all the available data off the network, it will wait the **Active Timeout** period of time before closing the connection.

Maximum Memory per Request:

The maximum amount of memory that the Process will allocate to each request, not including the HTTP body.

The same fields must be completed for the HTTPS Interface (on the **Network** tab) as for the HTTP Interface, with the addition of the **X.509 Certificate**, **Ciphers**, and **SSL session cache size** configuration options.

Click the **X.509 Certificate** button to select the certificate that the SOA Security Gateway will use to authenticate itself to clients during the SSL handshake. The list of certificates currently stored in the **Certificate Store** will be presented, from which a single certificate must be selected.

The **Ciphers** field allows the administrator to specify the ciphers that the server supports. The server will select the highest strength cipher (that is also supported by the client) from this list as part of the SSL handshake. Please refer to the OpenSSL documentation here [http://www.openssl.org/docs/apps/ciphers.html] for more information on the syntax of this setting.

The **SSL session cache size** field can be used to configure the maximum number of SSL sessions that this HTTPS interface can handle simultaneously.

Clients can be configured to authenticate to the SOA Security Gateway on the **Mutual Authentication** tab. There are 3 options available on this tab:

1. **Ignore Client Certificates:**

   The SOA Security Gateway will ignore client certificates if they are presented during the SSL handshake.

2. **Accept Client Certificates:**

   Client certificates will be accepted when presented to the SOA Security Gateway, but clients that do not present certificates will not be rejected.

3. **Require Client Certificates:**

   The SOA Security Gateway will only accept connections from clients that present a certificate during the SSL handshake.

Client certificates are typically issued by a Certificate Authority (CA). In most cases, the CA will include a copy of its certificate in the client certificate so that consumers of the certificate can decide whether or not to trust the client based on the issuer of the certificate.

It is also possible that a *chain* of CAs were involved in issuing the client certificate. For example, a top-level organization-wide CA (e.g. Company CA) may have issued department-wide CAs (e.g. Sales CA, QA CA, etc), and each department CA would then be responsible for issuing all department members with a client certificate.

In such cases, it is possible that the client certificate will contain a *chain* of 1 or more CA certificates. The **Maximum depth of client certificate chain** field on the **Mutual Authentication** tab can be used to configure how many CA certificates in a chain of 1 or more CA certificates present in a client certificate will be trusted when validating the client certificate.

By default, only 1 issuing CA certificate is used, and this certificate must be checked in the list of "trusted root certificates". If more than 1 certificate is to be used, then only the top-level CA must be considered trusted, while the intermediate CA certificates are not.

Finally, select the root CA (Certificate Authority) certificates that the SOA Security Gateway will consider trusted when authenticating clients during the SSL handshake. Only certificates signed by the CAs selected here will be accepted.

Configuring Conditions for a HTTP Interface

It is possible to configure the SOA Security Gateway to bring down an active HTTP Interface if certain *conditions* fail to hold. For example, the HTTP Interface can be brought down if a Remote Host is not available or if a physical network interface on the machine on which the SOA Security Gateway is running loses connectivity to the network.

For more information on configuring *conditions* for HTTP Interfaces, take a look at the Configuring Conditions for HTTP Interfaces help page.

Relative Paths

The Relative Path binds a policy to a specific relative path. When the SOA Security Gateway receives a request on this relative path, it will invoke the specified policy. Take a look at A First Tutorial: Protecting the Sample Web Service for more information on how to configure policies.

To configure a Relative Path for a given Service Group, right-click on the Service Group in the tree view and select the **Add Relative Path** option from the drop down menu. Relative Paths are added using the **Configure Relative Path** dialog. Complete the following fields on this dialog:

Relative Path:

Requests received on this relative path will be processed by the policy selected in the **Policy** field.

Policy:

The SOA Security Gateway will invoke the policy selected here when it receives a request on the path specified in the **Relative Path** field.

Static Content Provider

A *Static Content Provider* can be used in conjunction with a HTTP Interface to serve static content from a specified directory. A relative path is associated with each Static Content Provider so that requests received on this path will be dispatched directly to the Provider and will not be mapped to a Policy in the usual manner.

So, for example, it is possible to configure a Static Content Provider to serve content from the "c:\docs" folder (on a Windows system) when it receives requests on the relative path "/docs".

To add a Static Content Provider, right-click on the Service Group under the Process and select the **Static Content Provider** menu option. Complete the following fields on the **Static Content Provider** dialog:

Relative Path:

Enter the path that you want to receive requests for static content on.

Content Directory:

Enter or browse to the location of the directory that you want to serve static content from.

Index File:

Enter the name of the file that you want to use as the index file for content retrieved from the directory specified in the field above. This file will be retrieved by default if no resource is explicitly specified in the request URI. So, for example, if the client requests **http://[HOST]:8080/docs/** (i.e. with only a relative path specified as opposed to a specific resource), the file specified here will be retrieved. This file must exist in the directory specified in the previous field.

Allow Directory Listings:

If this option is checked, the Static Content Provider will return a full directory listings for requests specifying a relative path only. So, for example, if this option is checked and if a request is received for **http://[HOST]:8080/docs/samples**, the list of directories under this directory will be returned, assuming that this directory exsits on the file system. This option can be turned off to prevent attacks where a hacker can send up different request URIs in the hope that the server will return some information about the directory structure of the server.

Servlet Applications

Developers may wish to write their own Java servlets and deploy them under the SOA Security Gateway in order to server HTTP traffic. Conversely, they may want to remove some of the default servlets from the out-of-the-box configuration, for example, to remove the ability to view logging remotely. This pairing down of unwanted functionality may be required to further lock down the box on which the SOA Security Gateway is running.

Note: Adding and removing Servlet Applications should only be carried out by developers with very specific requirements and under strict guidance from the CA support

team. The information given below simply outlines how to configure the fields on the various dialogs used in setting up Servlet Applications. For more detailed instructions, please contact the CA Support Team.

There are a few default Servlet Applications available under the "Management Services" group. The table below shows what these default servlets are used for:

Servlet Application	*Use*
/admin/	Used for serving logging information to a remote viewer.
/runtime/	Used for refreshing the server after an update has been made to a policy.
/monitoring/	Used to display real-time monitoring of server transactions

Please note that deleting any of these Servlet Applications may prevent the SOA Security Gateway from functioning correctly. Default Servlet Applications should only be deleted under strict supervision of the CA Support Team.

To add a new Servlet Application, right-click on the Services Group that you want to add the servlet to and select the **Add Servlet Application** option. Configure the following fields on the **Servlet Application** dialog:

Relative Path:

Enter the servlet *context* in this field. Multiple servlets can be added under this context, where each servlet is mapped to a unique URI.

Session Timeout:

Enter the timeout in seconds after which an inactive session will be closed. Click on the **OK** button.

The new Servlet Application will now appear in the tree view of the SOA Security Gateway Management Console. To add a new servlet, right-click on the new Servlet Application and select the **Add Servlet** menu option. Configure the following fields on the **Servlet** dialog:

URI:

The path entered here maps incoming requests on a particular request URI to the Java servlet class entered in the field below. This path must be unique across all Servlets that are added under this Servlet Application (i.e. servlet context).

Class:

Enter the fully qualified class name of the servlet class. This class can be added to the server runtime by adding the Jar, class file, or package hierarchy to the **[VINSTDIR]/ext/lib** folder, where VINSTDIR points to the root of your SOA Security Gateway installation.

Read Timeout:

Specify the time in seconds that the servlet should wait before closing an idle connection.

Servlet Properties:

Properties can be configured for each servlet by clicking on the **Add** button and then entering a name and value in the fields provided on the **Properties** dialog.

Packet Sniffers

Packet Sniffers are a type of Passive Service. Rather than opening up a TCP port and *actively* listening for requests, the Packet Sniffer *passively* reads raw data packets off the network interface. The Sniffer assembles these packets into complete messages that can then be passed into an associated policy.

Because the Packet Sniffer operates passively (i.e. does not listen on a TCP port) and, therefore, completely transparently to the client, it is most useful for monitoring and managing Web Services. For example, the Sniffer can be deployed on a machine running a Web Server acting as a container for Web Services. Assuming that the Web Server is listening on TCP port 80 for traffic, the Packet Sniffer can be configured to read all packets destined for port 80 (or any other port, if necessary). The packets can then be marshalled into complete HTTP/SOAP messages by the Sniffer and passed into a policy that logs the message to a database, for example.

Important Note:

Please note that on Linux and Solaris platforms, the SOA Security Gateway must be started by the root user in order to gain access to the raw packets.

Since Packet Sniffers are mainly for use as passive monitoring agents, they are usually created within their own Service Group. So, for example, a new group can be created for this purpose by right-clicking on the Process, selecting the **Add Service Group** menu option, and the entering "Packet Sniffer Group" on the **Add Service Group** dialog.

We can then add a Relative Path Service to this Group by right-clicking on the "Packet Sniffer Group" and selecting the **Add Relative Path** menu option. Enter a path in the field provided and select the policy that you want to dispatch messages to when the Packet Sniffer detects a request for this path (after it assembles the packets). So, for example, if the Relative Path is configured as "/a", and the Packet Sniffer assembles packets into a request for this path, the request will be dispatched to the policy selected in the Relative Path Service.

Finally, we can add the Packet Sniffer itself by right-clicking on the "Packet Sniffer Group" node, selecting **Packet Sniffer**, and then the **Add** menu option. Complete the following fields on the **Packet Sniffer** dialog:

Device to Monitor:

Enter the name or identifier of the network interface that the Packet Sniffer will monitor. The default entry here is "any", but it is important to note that this is only valid on Linux. On UNIX-based systems, network interfaces are usually identified using names like "eth0", "eth1", and so on. On Windows, these names are more complicated, for example, "\Device\NPF_{00B756E0-518A-4144 ... }.

Filter:

The Packet Sniffer can be configured to only intercept certain types of packets. For example, it can ignore all UDP packets, only intercept packets destined for port 80 on the network interface, ignore packets from a certain IP address, listen for all packets on the network, and so on.

The Packet Sniffer uses the **libpcap** library filter language to achieve this. This language has a complicated but powerful syntax that allows you to *filter* what packets are intercepted and what packets are ignored. As a general rule, the syntax consists of one or more expressions combined with conjunctions, such as "and", "or", and "not". The following table lists a few examples of common filters and explains what they actually filter:

Filter Expression	*What does it filter?*
port 80	Capture only traffic for the HTTP Port (i.e. 80).
host 192.168.0.1	Capture traffic to and from IP address 192.168.0.1.
tcp	Capture only TCP traffic.
host 192.168.0.1 and port 80	Capture traffic to and from port 80 on IP address 192.168.0.1.
tcp portrange 8080-8090	Capture all TCP traffic destined for ports from 8080 through to 8090.

tcp port 8080 and not src host 192.168.0.1	Capture all TCP traffic destined for port 8080 but not from IP address 192.168.0.1.

The default filter of "tcp" simply captures all TCP packets arriving on the network interface. For more information on how to configure filter expressions like these, please refer to the man pages of **tcpdump** man page, which can be found here [http://www.tcpdump.org/tcpdump_man.html].

Promiscuous Mode:

When listening in "promiscuous mode", the Packet Sniffer will capture all packets on the same Ethernet network, regardless of whether or not the packets are addressed to the network interface that the Sniffer is monitoring.

Users

Overview

The purpose of this tutorial is to introduce the reader to the concept of a user in the SOA Security Gateway. We will examine the different types of **Users**, and look at how these users can be given permissions to perform special privileged operations.

Users

Users play a number of roles in the SOA Security Gateway. The typical user signs and sends SOAP messages to the SOA Security Gateway. However, users can be assigned extra privileges to effectively give them an administrator-type role. For example, a user can be assigned any or all of the following privileges, each of which will be described later in this tutorial:

- Sign XML messages
- Remote logging
- View reports

Adding Users

To view all the existing **Users**, click on the **UserStore** item in the tree view on the left-hand side of the SOA Security Gateway Management Console. The users are listed in a table in the main panel of the SOA Security Gateway Management Console.

Users can be created and imported into the SOA Security Gateway keystore using the **Users** interface on the SOA Security Gateway Management Console. Privileges can then be assigned to users through this same interface. Click the **Add** button on the **Users** page to view the **Add User** dialog.

There are two tabs on this dialog:

- User Details
- User Attributes

User Details:

To add a new **User**, complete the following fields:

1. **Name:**

   Enter a name for the new user.

2. **Realm:**

   This is a read-only field. All users are automatically added to the "ca.com" domain.

3. **User's Password:**

   Enter a password for this user here. If a hash of the password is to be used instead of a cleartext version, check the **MD5 Hash** checkbox.

4. **Confirm User's Password:**

   Re-enter the user's password to confirm it.

5. **MD5 Hash Password:**

   Select this checkbox if a hash of the user's password is to be stored.

6. **X.509 Cert:**

   Click the **X.509 Cert** button to load the user's certificate from the **Certificate Store**.

It is now possible to assign privileges to this new user. The following privileges can be assigned:

- **Remote Logging:**

  A remote Monitoring Console can be installed with the SOA Security Gateway, which enables server activity to be monitored from a remote machine. Users must have the **Remote Logging** privilege assigned to them in order to run the Monitoring Console remotely.

- **View Reports:**

  The SOA Security Gateway includes a browser-based reporting module that can be used to generate reports on server transactions. Users can only run reports if they have been granted the **View Reports** privilege. This is the only privilege which does not require the presence of the user's private key in the SOA Security Gateway keystore. This is because users can also use local keys which are stored in the

browser's keystore.

- **Sign XML Messages:**

  The SOA Security Gateway can sign outbound XML messages using the signing key of a **User**. This user must have been granted the **Sign XML Messages** privilege.

- **Sign Log Events:**

  The SOA Security Gateway maintains detailed logging information about all server security events. It is possible to sign these logs, thus providing a signed audit trail for all processed messages. Only users with the **Sign Log Events** privilege can be used to sign the logs.

- **Sign OCSP or XKMS Requests:**

  OCSP (Online Certificate Status Protocol) and XKMS (XML Key Management Specification) are both methods of certificate validation. A client presents a certificate to an OCSP/XKMS responder to determine whether or not the certificate is valid or not. Such requests must be signed, and the SOA Security Gateway will only allow users which have been assigned the **Sign OCSP or XKMS Request** privilege to sign these requests.

- **Use for Client Authentication:**

  Whenever the SOA Security Gateway needs to authenticate to another service (for example, over two-way SSL to an LDAP directory), it needs to use a client certificate. The user whose certificate is to be used must be assigned the **Use for Client-side SSL Authentication** privilege.

User Attributes:

This section allows you to configure user attributes as simple name-value pairs. The following are examples of user attributes:

- role=admin
- email=niall@ca.com
- dept=eng
- company=ca

User attributes can be added by clicking the **Add** button. Enter the name of the attribute in the **Name** field. Enter the value of the value of the attribute in the **Value** field.

Certificate Store

Overview

In order for the SOA Security Gateway to trust X.509 certificates issued by a particular CA (Certificate Authority), it is necessary to import that CA's certificate into the SOA Security Gateway's Trusted Certificate Store. For example, if the SOA Security Gateway is to trust secure communications (i.e. SSL connections or XML Signature) from an external SAML PDP (Policy Decision Point), it would be necessary to import either the PDP's certificate itself, or the issuing CA's certificate into the SOA Security Gateway.

Configuration

To view the list of certificates stored in the Certificate Store, click on the **Certificates** item in the tree view on the left-hand side of the SOA Security Gateway Management Console. The certificates are listed in a table in the main panel of the SOA Security Gateway Management Console.

To create a certificate and private key, click the **Create** button on the **Certificates** screen of the **SOA Security Gateway Management Console**. The **Configure Certificate and Private Key** dialog is displayed.

X.509 Certificate Tab:

The following configuration options are available on this tab:

- **Subject:**

  Click the **Edit** button to configure the *distinguished name* of the subject.

- **Public Key:**

  Click the **Import** button to import the subject's public key (usually from a PEM- or DER-encoded file).

- **Version:**

  This read-only field displays the X.509 version of the certificate.

- **Issuer:**

  This read-only field displays the distinguished name of the CA that issued the certi-

ficate.

- **Validity Period:**

  The dates specified here define the validity period of the certificate.

- **Alias Name:**

  This mandatory field allows the user to specify a friendly name (or alias) for the certificate.

- **Use Distinguished Name:**

  Check this option to view the DName of the certificate in the text box instead of the certificate alias.

- **Import Certificate:**

  Click this button to import a certificate from a file.

- **Export Certificate:**

  Use this option to export the certificate to a file.

- **Sign Certificate:**

  Click this button to sign the certificate. The certificate can either be self-signed, or it can be signed by the private key belonging to a trusted CA whose key pair has been stored in the **Certificate Store**.

Private Key Tab:

Use the **Private Key** tab to configure details of the private key. By default, private keys are stored locally in the Certificate Store. They can also be stored on a HSM (Hardware Security Module), if required.

Private Key Stored Locally:

Select the **Private key stored locally** radio button. The following configuration options are available for keys that are stored locally in the Certificate Store:

- **Private Key:**

This read-only field displays details of the private key.

- **Import Private Key:**

  Click the **Import Private Key** button to import the subject's private key (usually from a PEM- or DER-encoded file).

- **Export Private Key:**

  Click this button to export the subject's private key to a PEM- or DER-encoded file.

Private key stored on HSM:

If the private key that corresponds to the public key stored in the certificate resides on a HSM, you should select the **Private key stored on HSM** radio button. Configure the following fields to associate a key stored on a HSM with the current certificate:

- **Engine Name:**

  Enter the name of the OpenSSL Engine to use to interface to the HSM. All vendor implementations of the OpenSSL Engine API are identified by a unique name. Please refer to your vendor's HSM or OpenSSL Engine implementation documentation to find out the name of the engine.

- **Key ID:**

  The value entered here will be used to uniquely identify a specific private key from all others that may be stored on the HSM. On completion of the dialog, this private key will be associated with the certificate that you are currently editing.

Global Options:

The following global configuration options apply to both the **X.509 Certificate** and **Private Key** tabs:

- **Import Certificate + Key:**

  Use this option to import a certificate and a key from a file.

- **Export Certificate + Key:**

  Use this option to export a certificate and a key to a file.

Click the **OK** button when you have finished configuring the certificate and/or private key.

Back on the main **Certificates** screen, it is possible to edit an existing certificate using the **Edit** button. It is also possible to just view the details of an existing certificate using the **View** button. Similarly, a certificate can be removed from the Certificate Store using the **Remove** button.

It is also possible to export a certificate to a Java keystore. This can be done by selecting the certificate in the table and then clicking on the **Export to Keystore** button. Choose the name and location of the new keystore file and enter a passphrase for this keystore when prompted.

System Alerting

Overview

The purpose of this tutorial is to demonstrate SOA Security Gateway's enhanced logging capabilities. Usually alerts are sent when a filter fails, but can also be used for notification purposes. SOA Security Gateway can send system alerts to several alert destinations, including the Unix syslog, Windows Event Log, and an email recipient.

There are 2 steps involved in configuring SOA Security Gateway to send system alerts:

1. Configure an alert destination
2. Configure a **System Alert** filter

Alert Destinations

The first step in configuring SOA Security Gateway to send alerts is to configure an *alert destination*. SOA Security Gateway can send alerts to the following destinations.

- Unix Syslog
- Windows Event Log
- CheckPoint's FireWall-1
- Email Recipient

Unix Syslog

Many types of Unix provide a general purpose logging utility called *syslog*. Both local and remote processes can send logging messages to a centralized system logging daemon, known as *syslogd*, which in turn writes the messages to the appropriate log files. It is possible to configure the level of detail at which *syslog* logs information. This allows administrators to centrally manage how log files are handled, rather than separately configuring logging for each process.

Each type of process logs to a different syslog *facility*. There are facilities for the kernel, user processes, authorization processes, daemons, as well as a number of place-holders which can be used by site-specific processes. SOA Security Gateway allows you to log to a number of facilities, including *auth*, *daemon*, *ftp*, *local0-7*, and *syslog* itself.

To configure a Unix syslog alert destination, right click on **Alerts** in the tree view on the left-hand side of the **SOA Security Gateway Management Console**, and select the **Add** option, followed by the **Syslog Alert** option.

The **Syslog Alerting** dialog allows you to specify details about the machine on which the syslog daemon is running. SOA Security Gateway will connect to this daemon and log to the specified facility when the alert event is triggered.

Complete the following fields on the **System Alerting** dialog:

- **Name:**

  Enter a name for this alert destination.

- **Host:**

  The host name or IP address of the machine where the syslog daemon is running.

- **Facility:**

  The facility that SOA Security Gateway should send alerts to.

Windows Event Log

This alert destination allows alert messages to be written to the local or a remote Windows Event Log. To add a Windows Event Log alert destination, right click on **Alerts** in the tree view on the left-hand side of the **SOA Security Gateway Management Console**, and select the **Add** option, followed by the **Windows Event Log** option.

The **Windows Event Log Alerting** dialog allows you to specify the machine whose Event Log SOA Security Gateway will send alerts to.

Complete the following fields on this dialog:

- **Name:**

  Enter a name for this alert destination.

- **UNC Server name:**

  Enter the UNC (Universal Naming Code) of the machine where the event log resides. For example, to send alerts to the event log running on a machine called NT_SERVER, enter "\\NT_SERVER" as the UNC name for this host.

CheckPoint's FireWall-1

SOA Security Gateway is OPSEC compliant. OPSEC compliancy is awarded by Check-Point Software Technologies to products that have been successfully integrated with at least one of their products. In this case, SOA Security Gateway has been integrated with CheckPoint's FireWall-1 product.

FireWall-1 is the industry leading firewall that provides network security based on a security policy created by an administrator. Although OPSEC is not an open standard, the platform is recognized worldwide as the standard for interoperability with regards to network security and the alliance contains over 300 different companies. OPSEC integration is achieved through a number of published application programming interfaces (APIs), which enable third-party vendors to interoperate with CheckPoint's products.

SOA Security Gateway connects to an OPSEC compliant firewall and prevents further requests for the particular client that triggered the alert.

To configure a Unix syslog alert destination, right click on **Alerts** in the tree view on the left-hand side of the **SOA Security Gateway Management Console**, and select the **Add** option, followed by the **OPSEC Alert** option.

The **OPSEC Alerting** dialog allows you to specify details about the machine on which FireWall-1 is installed, the port it is listening on, as well as how to authenticate to the firewall. SOA Security Gateway will connect to the specified firewall when the alert event is triggered and prevents further requests for the particular client that triggered the alert

Configuration information can be stored in a file and then loaded using the **Browse** button. Alternatively, the **Template** button can be used to simply load the required settings into the text area. The configuration values can then be added manually. Either way, the following configuration settings must be set:

- **sam_server auth_port:**

  The port number to be used for establishing SIC (Secure Internal Communications) based connections with the firewall.

- **sam_server auth_type:**

  The authentication method to use when connecting to the firewall.

- **sam_server ip:**

  The host name or IP address of the machine that is hosting the Checkpoint Firewall.

- **sam_server opsec_entity_sic_name:**

  The firewall's SIC name.

- **opsec_sic_name:**

  The OPSEC application's SIC Name, which is just the application's full DName as defined by the VPN-1 SmartCenter Server.

- **opsec_sslca_file:**

  The name of the file containing the OPSEC application's digital certificate.

For more information on these settings, please consult the documentation for your OPSEC application.

In order for SOA Security Gateway to establish the SSL connection to the firewall, the `opsec_sslca_file` specified above must be uploaded to the SOA Security Gateway machine. This can be done by clicking the **Add** button at the bottom of the screen.

Email Recipient

This alert destination allows alert messages to be sent by email. To add a Windows Event Log alert destination, right click on **Alerts** in the tree view on the left-hand side of the **SOA Security Gateway Management Console**, and select the **Add** option, followed by the **Email Alert** option.

The **Email Alerting** dialog allows you to configure how the email alert is to be sent. Complete the following fields on this dialog:

- **Name:**

  Enter a name for this alert destination.

- **Email Recipient (To):**

  Enter the recipient of the alert mail in this field. Use a comma-separated list of email addresses to send alerts to multiple recipients.

- **Email Sender (From):**

  Email alerts will appear *from* the sender email address specified here. It is important to note that some mail servers do not allow relaying mail where the sender in

the "from" field is unrecognized by the server.

- **Outgoing Mail Server (SMTP):**

  Specify the SMTP server that SOA Security Gateway should use to relay the alert email.

- **User Name:**

  If you are required to authenticate to the SMTP server, enter the username to use for authentication in this field.

- **Password:**

  Enter the password for the user name specified above.

- **Email Debugging:**

  Email debugging can be used to find out more information about errors encountered by SOA Security Gateway when attempting to send email alerts. All trace files are written to the `/trace` directory of your SOA Security Gateway installation.

System Alert Filter

Typically, an **Alert** filter is placed on the failure path of another filter in the policy. For example, it is possible to configure an alert if a schema validation fails 10 times within a 5000 millisecond period for a specified Web Service. In this case, you would place the **Alert filter** on the failure path from the **Schema Validation** filter. This situation is illustrated in the following policy snapshot:

The **Alert** filter itself can be located in the **Monitoring** filter group. Configure the following fields on the **Alert Filter** screen:

- **Name:**

  Enter a descriptive name for the filter here.

- **Accumulated number of messages:**

  Enter the number of times this filter can be invoked before the alert will be sent.

- **In time period (secs):**

Enter the time period in which the accumulated number of messages can occur before and alert is triggered.

- **Track per client:**

  Check this option if you want to record the accumulated number of messages in the specified time period for each client.

- **Alert Type:**

  Select the severity level of the alert. This option is only relevant for alert destinations that support severity levels, such as the Windows Event Log.

- **Alert Message:**

  Enter the text that will appear in the alert in this field. It is possible to enter message attributes as wildcards in this field. For example, instead of sending a generic alert stating that, "Authentication Failed", it is possible to use a message attribute to give the ID of the user that was not authenticated. To do this, use the following syntax:

  ```
  ${name_of_attribute}
  ```

  The following examples show how to use message attributes in alert messages:

  - "Authentication failure for user: ${authentication.subject.id}".
  - "${alert.number.failures} authentication failures have occurred in ${alert.time.period} seconds."
  - "${alert.number.failures} exceptions have occurred in circuit ${circuit.name}."
  - "The last exception was ${circuit.exception} with path ${circuit.path}."

  Note that an alert message is not required for alerts to an OPSEC firewall.

- **Alert To:**

  All pre-configured alert destinations are displayed in the **Alert To** table. Select the destination(s) that SOA Security Gateway will send alerts to if the criteria specified for this filter are met.

References

OPSEC [http://www.opsec.com]

Check Point Software Technologies [http://www.checkpoint.com]

Global Caches

Overview

In cases where a Web Service is serving the same request (and generating the same response) over and over again, it makes sense to use a caching mechanism. When a cache is employed, a unique identifier for the request is cached together with the corresponding response for this request. If an identical request is received, the response can be retrieved from the cache instead of forcing the Web Service to re-process the identical request and generate the same response. The use of caching in this way helps divert unnecessary traffic from the Web Service and makes it more responsive to new requests.

For example, let's assume we have deployed a Web Service that returns a list of cities in the USA from an external database, which is then used by a variety of Web-based applications. Since the names and quantity of cities in the USA is fairly constant, it is fair to say that if the Web Service is handling hundreds or thousands of requests every day, this represents a serious waste of processing time and effort - especially considering that the database that contains the relatively fixed list of city names is hosted on a separate machine to the service.

If we assume that the list of cities in the database does not change very often, it makes sense to use CA SOA Security Gateway to cache the response from the Web Service that contains the list of cities. So then, when a request for this particular Web Service is identified by the SOA Security Gateway, the cached response can be returned to the client. This approach results in the following performance improvements:

- The SOA Security Gateway does not have to route the message on to the Web Service, therefore saving the processing effort required to this, and perhaps, more importantly, saving the time it takes for the round trip.
- The Web Service does not have to waste processing power on generating the same list over and over again, therefore making it more responsive to requests for other services.
- Assuming a naive implementation of database retrieval and caching, the Web Service does not have to query the database (over the network) and collate the results over and over again for every request.

The caching mechanism used within the SOA Security Gateway offers full control over the size of the cache, the lifetime of objects within the cache, whether objects are cached to disk or not, and even, whether caches can be replicated across multiple instances of SOA Security Gateways. The following sections describe how to configure

both local and distributed caches within the SOA Security Gateway, concluding with a detailed example of how to configure a circuit to cache responses.

Local Caches

Local caches are used where a single SOA Security Gateway instance has been deployed. In such cases, it is not necessary to replicate caches across multiple running instances of the SOA Security Gateway.

A local cache can be added by selecting the top-level **Caches** tree item in the SOA Security Gateway Management Console and clicking the **Add** button at the bottom of the screen. Select the **Add Local Cache** option from the menu. The following fields can be configured on the **Configure Local Cache** dialog:

Cache Name:

Enter a name for the cache.

Maximum Elements in Memory:

Enter the maximum number of objects that can be in memory at any one time.

Maximum Elements on Disk:

This field sets the maximum number of objects that can be stored in the disk store at any one time. A value of zero indicates an unlimited number of objects.

Eternal:

If this option is checked, objects stored in the caches will never expire and timeouts will have no effect.

Overflow to Disk:

Check this option if you want the cache to overflow to disk when the number of objects in memory has reached the amount set in the **Maximum Elements in Memory** field above.

Note: The following fields are optional:

Time to Idle:

This field determines the maximum amount of time (in seconds) between accesses that an object can remain idle before it expires. A value of zero indicates that objects can idle for infinity, which is the default value. Note that if the **Eternal** field above is

checked, this setting will be ignored.

Time to Live:

Sets the maximum time between when an object is created and when it expires. The default value is zero, which means that the object can live for infinity. Note that if the **Eternal** field above is checked, this setting will be ignored.

Persist to Disk:

If this field is checked, the disk store will be persisted between JVM restarts. This option is turned off by default.

Disk Expiry Interval:

This setting configures the number of seconds between runs of the disk expiry thread. The default value is 120 seconds.

Disk Spool Buffer Size:

This setting indicates the size of memory (in MBs) to allocate the disk store for a spool buffer. Writes are made to this memory and then asynchronously written to disk. The default size is 30MB, however if you get OutOfMemory exceptions, you may consider lowering this value. On the other hand if you notice poor performance, you should increase the value.

Eviction Policy:

Select the eviction policy that the cache will use to evict objects from the cache. The default policy is "Least Recently Used", however it is also possible to use "First in First Out" and "Less Frequently Used".

Distributed Caches

If you have deployed several SOA Security Gateways throughout your network, you will need to employ a distributed cache. In this scenario, each SOA Security Gateway has its own local copy of the cache but registers a cache event listener that replicates messages to the other caches so that put, remove, expiry, and delete events on a single cache are duplicated across all other caches.

A distributed cache can be configured by clicking on the **Caches** top-level tree item in the SOA Security Gateway Management Console and then clicking the **Add** button on the bottom of the screen. Select the **Add Distributed Cache** menu option and configure the following fields on the **Configure Distributed Cache** dialog:

Note: A lot of the settings for the distributed cache are identical to those for the local

cache. Please refer to the Local Caches section above for more information on how to configure these fields. The information below only refers to fields that do not appear on both dialogs.

Event Listener Class Name:

Enter the name of the listener factory class that enables this cache to register listeners for cache events, such as put, remove, delete, and expire.

Properties Seperator:

Specify the character to use to separate the list of properties in the field below.

Properties:

This field can be used to list the properties to pass to the RMICacheReplicatorFactory. The following properties are available:

- **replicatePuts=true | false**

  Determines whether new elements placed in a cache are replicated to other caches. Default is true.

- **replicateUpdates=true | false**

  Determines whether new elements that override (i.e. update) existing elements with the same key in a cache are replicated. Default is true.

- **replicateRemovals=true**

  Determines whether element removals are replicated. Default is true.

- **replicateAsynchronously=true | false**

  Determines whether replications are asynchronous (true) or synchronous (false). Default is true.

- **replicateUpdatesViaCopy=true | false**

  Determines whether new elements are copied to other caches (true) or a remove message is sent (false). Default is true.

- **asynchronousReplicationIntervalMillis=[number of ms]**

The asynchronous replicator runs at a set interval of milliseconds. The default is 1000 and the minimum is 10. This property is only applicable if replicateAsynchronously=true.

Cache Bootstrap Class Name:

Specifies a BootstrapCacheLoader factory that the cache can call on initialization to prepopulate itself. The RMIBootstrapCacheLoader bootstraps caches in clusters where RMICacheReplicators are used.

Properties Seperator:

The character entered here is used to separate the list of properties listed in the field below.

Properties:

The properties listed here will be used to initialize the RMIBootstrapCacheLoaderFactory. The following properties are recognized:

- **bootstrapAsynchronously=true | false**

  Determines whether the bootstrap happens in the background after the cache has started (true), or if bootstrapping must complete before the cache is made available (false). Default is true.

- **maximumChunkSizeBytes=[integer]**

  Caches can potentially grow larger than the memory limits on the JVM. This property allows the bootstrapper to fetch elements in chunks. The default chunk size is 5000000 (i.e. 5MB).

Cache Settings

In a distributed cache, there is no master cache controlling all caches in the group. Instead each cache is a peer in the group and needs to know where all the other peers in the group are located. *Peer Discovery* and *Peer Listeners* are 2 essential parts of any distributed cache system.

They can be configured by right-clicking on the **Caches** top-level node in the tree view of the SOA Security Gateway Management Console and selecting the **Edit Cache Settings** menu option. Configure the following fields on the **Cache Settings** dialog:

Peer Provider Class:

By default, the built-in peer discovery class factory `net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory` is used.

Properties Separator:

Specify the token that will be used as the separator for the list of properties in the next field.

Properties:

The properties listed here dictate whether or not the peer discovery mechanism is automatic or manual. If the automatic mechanism is used, each peer uses TCP multicast to establish and maintain a multicast group. This is the default option since it requires minimal configuration and peers can be automatically added and removed from the group. Each peer pings the group every second. If a peer has not pinged any of the other peers after 5 seconds it is dropped from the group, while a new peer is admitted to the group if it starts pinging the other peers. To use automatic peer discovery ensure that the `peerDiscovery` setting is set to "automatic". You can specify the multicast address and port using the `multicastGroupAddress` and `multicastGroupPort` settings. The time to live for multicast datagrams can be specified using the `timeToLive` setting.

Alternatively, it is possible to configure a manual peer discovery mechanism, whereby each peer definitively lists the peers that it wants to communicate with. This should only really be used in networks where there are problems propagating multicast datagrams. To use a manual peer discovery mechanism, make sure the `peerDiscovery` setting is set to "manual". The list of RMI URLs of the other peers in the group must also be specified, for example:

```
rmiUrls=//server2:40001/sampleCache1|//server2:40001/sampleCache2 .
```

Peer Listener Class:

The peer listener class specified here is responsible for listening for messages from peers in the group.

Properties Separator:

Specify the token that will be used to separate the list of properties in the field below.

Properties:

The properties entered here configure the way the listener behaves. Valid properties are

as follows:

- **hostname (optional)**

  This is the hostname of the machine on which the listener is listening. Note that, by default, this is set to "localhost", which maps to the local loopback address of 127.0.0.1, which is not addressable from another machine on the network. If you intend this cache to be used over the network, you should change this address to the IP address of the network interface on which the listener is listening.

- **port (mandatory)**

  Specify the port on which the listener is listening, which by default is 4001. This setting is mandatory.

- **socketTimeoutMillis (optional)**

  Enter the number of seconds that client sockets will wait when sending messages to this listener until they give up. The default setting is 2000ms.

Notify replicators of removal of items during refresh:

A server refresh will automatically purge all items from the cache. If this checkbox is enabled, the contents of each peer in the group will also be purged. This avoids a situation where a single peer is refreshed (and, therefore, has its contents purged) but the other peers in the group are not purged. If this option is not checked, the refreshed peer will attempt to bootstrap itself to the other peers in the group resulting in the cache items becoming replicated in the refreshed cache. This effectively negates the effect of the server refresh and may result in inconsistent behavior.

Example of Caching Response Messages

In this simple example, we will show how to construct a circuit that will cache responses from the Web Service. We will use the request body to identify identical successive requests. In other words, if the SOA Security Gateway receives 2 successive requests with an identical message body, it will return the corresponding response from the cache instead of routing the request to the Web Service.

The following diagram illustrates the complete circuit:

The logic of the circuit can be summarized as follows:

1. The purpose of the first filter is to configure what part of the request we want to use to identify unique requests. In this example, we have elected to use the request body as the unique key, which will then be used to lookup the appropriate response message from the cache.
2. The 2nd filter looks up the request body in the response cache to see if it contains the request body. If it does, the response message that corresponds to this request is returned to the client.
3. If it doesn't, the request is routed to the Web Service, which processes it (by connecting to a database over the network and running a SQL statement) and returns a response do the SOA Security Gateway.
4. The SOA Security Gateway then returns the response to the client and caches it in the response cache.
5. When the next identical request is received by the SOA Security Gateway, the corresponding response will be located in the responses cache and returned immediately to the client.

The following caching filters must be configured in order to achieve this circuit. For convenience, the routing filters will not be included since the configuration options here will depend on your target Web Service.

Create Key Filter:

This filter is used to decide what part of the request is used in order for a request to be considered unique. Different parts of the request can be identified internally using CA message attributes, for example, **content.body** contains the request body. The following fields must be configured for this filter:

- **Name:** Use request body to create unique key
- **Attribute Name:** content.body

Is Cached?:

This filter looks up the cache to see if a response has been stored for the current request. It looks up the cache using a message attribute, which is **message.key** by default. The **message.key** attribute contains a hash of the request message and can be used as the key for objects in the cache. If the key is found in the cache, the value of the key (i.e. the cached response for this request) is written to the **content.body** attribute, which can then be returned to the client using the **Reflect** filter. The following fields must be configured:

- **Name:** Have we already cached a response for this request?
- **Cache containing key:** Response Cache [Assuming we have created a cache of this name]
- **Attribute Containing Key:** message.key
- **Overwrite attribute name if found:** content.body

Reflect:

If the **Is Cached?** filter passes it retrieves the response from the cache and stores it in the **content.body** message attribute. The **Reflect** filter is then used to return the cached response to the client.

Routing:

If a response for this request could not be located in the cache, the SOA Security Gateway will route the request to the Web Service and wait for a response. Please refer to the Routing Configuration tutorial for more information on how to route messages.

Cache Attribute:

Once the response has been received from the Web Service it should be cached for future use. The **Cache Attribute** filter is used to configure the key that is used to lookup the cache and also what aspect of the response message is actually stored as the key value in the cache. Note that in this case, we specify the value of the **content.body** attribute to cache because since this filter is configured *after* the routing filters, this attribute will actually contain the response message. Note also that the value entered in the **Attribute Key** field should match that entered in the **Attribute containing key** field in the **Is Cached?** filter. The following fields should be configured:

- **Name:** Cache response body
- **Cache to use:** Response Cache
- **Attribute key:** message.key
- **Attribute name to store:** content.body

For more information on each of these filters, take a look at the following help pages:

Filter	***Help Page***
Create Key	Create Key

Filter	***Help Page***
Is Cached?	Is Cached?
Cache Attribute	Cache Attribute

Global Schema Cache

Overview

The **Schema Cache** contains XML Schemas that can then be used globally by **Schema Validation** filters. XML Schemas can be imported from either XML Schema files or WSDL files. WSDL files often contain XML Schemas that define the elements that appear within SOAP messages. To facilitate this, the SOA Security Gateway Management Console can import WSDL files from the file system, from a URL, or from a UDDI registry.

Once the XML Schema has been imported into the cache and selected in a **Schema Validation** filter, the SOA Security Gateway will retrieve the schema from the cache as opposed to fetching it from its original location. This improves the runtime performance of the filter, but also ensures that an administrator can have complete control over the Schemas that are used to validate messages.

The **Schema Cache** is available as a top-level node in the tree view of the SOA Security Gateway Management Console. The list of schemas present in the cache are listed in the **Imported Schemas** table. It is possible to view the contents of any of these schemas by simply clicking on the schema in the table. The schema contents will be displayed in the **Contents** text area.

At any stage, the contents of the schema can be manually modified in the text area. To save the modified contents to the cache, click the **Update** button.

Adding Schemas to the Cache

To add an XML Schema to the cache, click the **Add** button at the bottom of the **Schema Cache** screen. The **Load Schema** dialog offers 2 possible locations from which you can load a schema - from an XML Schema file directly or from a WSDL file.

Select the **From XML Schema** radio button to load the schema directly from a schema file and then click **Next**. On the next screen, enter or browse to the location of the schema file using the field provided. A full URL can also be entered here in order to pull the schema from a web location. Click the **Finish** button to import the schema into the cache.

Alternatively, if you want to load the schema file from a WSDL file, select the **From WSDL** radio button on the **Select Schema Source** screen and then click the **Next** button.

The WSDL file can be located from the file system, from a URL, or from a UDDI registry. Select the appropriate option and enter or browse to the location of the WSDL file in the fields provided. If you wish to retrieve the WSDL file from a UDDI registry, click the **WSDL from UDDI** radio button and then click the **Browse UDDI** button. The **Browse**

UDDI Server for for WSDL dialog allows you to connect to a UDDI and search it for a particular WSDL file. For more information on how to configure this dialog, please refer to the Retrieving WSDL Files from a UDDI Directory help page.

When importing the schema from the WSDL file it is also possible to check the WSDL file for compliance with the WSI Basic Profile. The Basic Profile consists of a set of assertions or guidelines on how to ensure maximum interoperability between different implementations of Web Services. For example, there are recommendations on what style of SOAP to use (e.g. "document/literal"), how schema information should and should not be included in WSDL files, and how message parts should be defined to avoid ambiguity for consumers of WSDL files.

The SOA Security Gateway Management Console can test WSDL files while extracting schemas from them for conformance against the recommendations laid out in the Profile. A report is generated showing exactly what recommendations have passed and what ones have failed. While a WSDL file that does not conform to the Profile can still be imported, there is no certainty that consumers of the Web Service will be able to use it without encountering problems.

Important Note:

In order to run the WSI compliance test, the WSI Test Assertions Document (TAD) file must be installed on the machine on which the SOA Security Gateway Management Console is running. This file can be obtained from www.ws-i.org [http://www.ws-i.org]. The *full path* to the location of this file (e.g. **c:/wsi/SSBP10_BP11_TAD.xml** must be specified in the global preferences for the SOA Security Gateway Management Console. To configure this setting, select the **Settings -> Preferences** option from the main menu bar of the SOA Security Gateway Management Console. In the **Settings** dialog, select the "WS-I TAD File" setting and then enter the location of the TAD file in the field provided.

To run the WSI compliance test on a WSDL, click the **Run WSI Compliance Test** button after selecting the WSDL file to import the schema from using the radio buttons on this screen. The results of the compliance test are displayed in the **WSI Compliance Results** dialog.

The overall result of the compliance test is given in the "Summary Result" column next to the top-level node, which reflects the name of the service. The individual results of the WSI compliance tests are grouped together by type beneath the top-level node. So, for example, all the test assertions that relate to the <wsdl:portType> elements in the WSDL are listed under the "portType" node, while all assertions relating to the <wsdl:operation> elements are listed under the "operation" node in the results tree.

Specific details about each assertion can be viewed in the **Assertion Details** section by clicking on the ID (e.g. "BP2103") of the assertion in the results tree. The details correspond to the information given for the assertion in the TAD file.

Schema Validation

The **Schema Validation** filter is used to validate XML messages against schemas stored either in the cache or in the **Web Services Repository**. The filter can be found in the Content Filtering category of filters in the SOA Security Gateway Management Console. For more information on configuring this filter, please refer to the help page for Schema Validation.

Messaging System

Overview

A *messaging system* is a loosely coupled, peer-to-peer facility where clients can send messages to, and receive messages from any other client. A messaging system differs from an email system in that the communication takes place between software components, as opposed to people, or people and software components.

In a messaging system, a client sends a message to a messaging agent. The recipient of the message can then connect to the same agent and read the message from it. However, the sender and recipient of the message do not have to be available at the same time in order to communicate (unlike HTTP, for example). The sender and recipient need only know the name and address of the messaging agent to talk to. In this way, messaging systems are said to be loosely coupled.

The Java Messaging System (JMS) is an implementation of such a messaging system. It provides an API for creating, sending, receiving, and reading messages. Java-based applications can use it to connect to other messaging system implementations.

A *JMS provider* can deliver messages synchronously and asynchronously, meaning that the client can either "fire and forget" messages or wait for a response before resuming processing. Furthermore, the JMS API can ensure different levels of reliability in terms of message delivery. For example, it can ensure that the message is delivered once and only once, or at least once.

The SOA Security Gateway uses the JMS API to connect to other messaging systems that expose a JMS interface, including IBM's MQSeries, JBoss Messaging, BEA's WebLogic Server, TIBCO's EMS, IBM's WebSphere Server, and SonicMQ from Sonic Software. As a consumer of a JMS queue or topic, the SOA Security Gateway can read XML messages and pass them into a policy for validation. These messages can then be routed onwards over HTTP or dropped on to another JMS queue or topic.

A JMS Wizard is available at the Process level, which can be used to configure the SOA Security Gateway to consume a JMS queue or topic. JMS Services can also be configured at the Process level, which can then be selected in the **Messaging System** Connection filter so that messages can be dropped on to the specified queue or topic.

The JMS Wizard and Service configurations are available by right-clicking on the Process in the tree view of the SOA Security Gateway Management Console and selecting the **Messaging System** menu option. The **JMS Wizard** and **JMS Service** options are available from this option.

Note: It is important to note that the JMS provider's jar files must be added to the SOA Security Gateway's classpath in order for this filter to function properly. To do this,

simply copy the provider's jar files to the `INSTALL_DIR/ext/lib` folder, where INSTALL_DIR points to the root of your SOA Security Gateway installation.

Configuring the JMS Service

The details entered in the **JMS Service** dialog are used by the SOA Security Gateway to drop messages on to a JMS queue or topic. The **Messaging System** Connection filter uses JMS Services configured here to do this.

Configure the following fields on the **JMS Service** dialog:

Name:

Enter a friendly name for the JMS Provider in the **Name** field.

Provider URL:

Enter the URL of the JMS provider, for example, **jnp://localhost:1099**.

Initial Context Factory:

The SOA Security Gateway uses a connection factory to create a connection with a JMS provider. A connection factory encapsulates a set of connection configuration parameters that have been defined by the administrator. For example, the initial context factory class for the JBoss application server is **org.jnp.interfaces.NamingContextFactory**.

Connection Factory:

Enter the name of the connection factory to use when connecting to the JMS provider. The name of the connection factory is vendor-specific. For example, the connection factory used for the JBoss application server is **org.jnp.interfaces:javax.jnp**.

Username:

If a username is required to connect to this JMS provider, enter it here.

Password:

Enter the password for this user.

Custom Message Properties:

JNDI context settings can be added by clicking on the **Add** button and adding name and value properties in the fields provided.

Session information can be configured for the newly added JMS Service by right-clicking

on it in the tree view in the SOA Security Gateway Management Console and selecting the **JMS Session** menu option. The **JMS Session** dialog consists of a single checkbox that determines whether or not to **Allow duplicates** for this session. By allowing duplicates, the SOA Security Gateway will delay sending acknowledgements for each JMS message it consumes for performance reasons, but will accept that duplicate messages may be sent by the producer while it is waiting for the SOA Security Gateway to acknowledge that it has consumed the message.

Since JMS is a reliable protocol, the consumer must send an acknowledgement for messages that it consumes. By toggling this option, you can configure whether the SOA Security Gateway sends acknowledgements for each message it receives, or whether it delays sending acknowledgements for performance reasons, but accepts that it may consume duplicate messages because the producer may not realize that the SOA Security Gateway is delaying the acknowledgements and may, therefore, believe that the SOA Security Gateway has not received the message, in which case it will attempt to re-send the message.

Once this JMS Service has been configured, you can configure the SOA Security Gateway to drop messages on to a queue or topic exposed by this Service by simply selecting it from the **JMS Session** field of the **Messaging System** Connection filter dialog when configuring a policy.

Configuring the JMS Wizard

The **JMS Wizard** is used to configure the Process to consume JMS messages from a JMS queue or topic. Once a message has been consumed by the SOA Security Gateway, it can be dispatched to a specified policy where the full compliment of message filters can be run on the message. The message can then be routed onwards over HTTP or dropped back on to a JMS queue or topic.

The **JMS Wizard** is available by right-clicking on the Process in the tree view of the SOA Security Gateway Management Console and selecting the **Messaging System** menu option. Select **JMS Wizard** from the menu.

Using the **JMS Wizard**, the first step is to configure details about the JMS provider, followed by JMS session information, and finally, the name of the queue or topic to consume is specified along with the policy to dispatch consumed messages to. Since the fields used to configure the JMS provider and session are the same as those used when configuring the **JMS Service**, please refer to the previous section for information on how to configure these fields. The remainder of this section will focus on the JMS consumer part of the configuration.

The following fields are available on the **JMS Consumer Configuration** screen of the wizard:

Auto MIME:

When this option is checked, the Process will automatically assume that the messages it consumes are MIME-formatted messages, for example, using SOAP over JMS. It should only be turned off if you are consuming raw XML data (or some other proprietary format) from the queue or topic. Please consult the CA support team for more information on how to do this.

Destination:

Enter the name of the queue or topic from which you want to consume JMS messages.

Selector:

The expression entered here is used to specify what messages the consumer is interested in receiving. By using a selector, the task of filtering the messages is actually performed by the JMS provider rather than by the consumer. The selector itself is a string that specifies an expression whose syntax is based on the SQL92 conditional expression syntax. By specifying a selector, the Process will only receive messages whose headers and properties match the selector.

Policy:

When the Process consumes messages from the JMS queue or topic selected in the fields above, it will dispatch them to the policy selected here. The policy processes the message and can then either route it onwards over HTTP or drop it back on a JMS queue or topic.

Directory Scanner

Overview

The Directory Scanner allows you to scan a certain directory on the file system for files containing XML messages. Once the messages have been read, they can be passed into the core message pipeline where the full collection of message processing filters can act on them.

The Directory Scanner is typically used in cases where an external application is dropping XML files (perhaps by FTP) on to the file system so that they can be validated, modified, and potentially routed onwards over HTTP, JMS, or simply stored back to another directory where the application can pick them up again.

This sort of protocol mediation is very useful in cases where legacy systems are involved. Instead of making drastic changes to the legacy system by adding a HTTP engine to it, for example, the SOA Security Gateway can be used to pull the files from the file system and then route them onwards over HTTP to another back-end system. The added benefit here, of course, is that messages are exposed to the full compliment of message processing filters made available by the SOA Security Gateway. This ensures that only properly validated messages will be routed on to the target system.

To add a new Directory Scanner, right-click on the name of the Process in the tree view of the SOA Security Gateway Management Console (e.g. "CA SOA Security Gateway") and select the **Directory Scanner -> Add** menu option. The following sections describe how to configure the fields on the **Directory Scanner Settings** dialog.

Directory to Scan

The fields configured in this section determine what files to scan, where to scan for them, and when to scan.

Name:

Enter or browse to the directory that the SOA Security Gateway will scan for XML files.

File Name Pattern:

If you only want to scan certain files based on some pattern, you can enter the pattern here. For example, if you only want to scan for files with a particular file extension, you can enter the file extension here. Similarly, if a particular naming scheme is used when dropping the files into the directory specified above, you can enter a pattern here to only scan for these files.

Poll Rate:

The poll rate entered here (in milliseconds) determines how often the SOA Security Gateway scans the directory for new XML files.

Directory for Output

Once the Directory Scanner has finished scanning the files it will move them to another directory to avoid processing them again. The fields configured in this section determine where processed files will be placed and what they will be called.

Name:

Enter the name of the directory to place the processed files. This may either be the response from the Web Service (after the policy has routed it onwards and received a response) or a modified message in cases where the policy has inserted a security token into the message or converted the message using an XSLT conversion, for example.

File Prefix:

If you would like to save processed files into the directory above with a prefix added to the filename, enter the prefix here. For example, you may want to prepend "_PROCESSED" to all processed files.

File Suffix:

Similarly you can add a suffix to the output files by entering the suffix in this field.

Completed Directory

Processed files will be removed from the source directory and placed into this directory post-processing to avoid re-processing the same files over and over again.

Working Directory

Files will be moved to the "temporary" directory specified here during processing. This is necessary in cases where the poll rate is quite low and there is a chance that the file may be scanned again before it is processed fully and moved to the completed directory.

Policy to Use

The messages contained within the scanned XML files will be passed into the policy selected here. If the policy routes messages to a Web Service, the response from the ser-

vice will be placed in the output directory specified above. Similarly, if the policy modifies the request by signing it or adding a security token to it, for example, the updated message will also be placed in the output directory.

External Connections

Overview

CA SOA Security Gateway can leverage your existing identity management infrastructure, thus avoiding the need to maintain separate "silos" of user information. For example, if you already have a database full of user credentials, the SOA Security Gateway can authenticate requests against this database, rather than using its own internal user store. Similarly, the SOA Security Gateway can authorize users, lookup user attributes, and validate certificates against 3rd party identity management servers.

Each such connection to an external system can be added as a global *External Connection* in the SOA Security Gateway Management Console so that it can be re-used across all filters and policies. So, for example, if you create a circuit that authenticates users against an LDAP directory and then validates an XML signature by retrieving a public key from the same LDAP directory, then it makes sense to create a global External Connection for that LDAP directory. The LDAP Connection can then be merely *selected* in both the authentication and XML signature verification filters, rather than having to be re-configured in both filters.

External Connections can also be used in cases where you want to configure a group of related URLs. This is most useful in cases where you want to round-robin between a number of related URLs to ensure high availability. When the SOA Security Gateway is configured to use a *URL Connection Set* (as opposed to just a single URL), it will round-robin between the URLs in the set.

In the following sections, we will look at each different type of External Connection and describe how to configure connections of that type.

Authentication Repository Profiles

As stated at the beginning of this guide, CA SOA Security Gateway can authenticate users against external databases and LDAP repositories, in addition to its own local user store. A number of bespoke authentication connectors are also available to allow the SOA Security Gateway to authenticate against specific 3rd party Identity Management products.

Connection details for these "authentication repositories" are configured at a global level, making them available for use across authentication (and authorization) filters. This saves the administrator from re-configuring connection details within each filter. We will now look at each type of authentication repository in turn.

Local Repositories:

The **Authentication Repository** can be maintained in the same local store as the SOA Security Gateway uses to store all its configuration information. To do this, complete the following steps:

- Right-click on the "Local Repositories" tree item in the SOA Security Gateway Management Console and select the **Add a New Repository** menu option.
- Enter a name for the repository in the **Repository Name** field.
- The **Authorization Attribute Format** field allows administrators to specify whether they want to use the client's **X.509 Distinguished Name** or **User Name** in subsequent Authorization filters. If "User Name" is selected, the user name used by the client to authenticate to the SOA Security Gateway will be used in any configured Authorization filters. If, on the other hand, "X.509 Distinguished Name" is selected, the X.509 DName stored by the SOA Security Gateway for that user will be used for subsequent authorization.

  If, for example, the administrator selected the "User Name" option from the **Authorization Attribute Format** dropdown, then "admin" (i.e. the **User Name** field) will be used for authorization.

  Alternatively, if the "X.509 Distinguished Name" menu option is selected, then the X.509 DName (e.g. "O=Company, OU=comp, EMAIL=emp@company.com, CN=emp") will be used for authorization.

Database Repositories:

The SOA Security Gateway can use a user profile repository that is stored in an external relational database. If an organization already has a "silo" of user profiles stored in the database and does not want to duplicate this store within the SOA Security Gateway's local configuration storage. Complete the following steps to configure the SOA Security Gateway to connect to an authentication repository stored in a database:

- Right-click on the "Database Repository" tree item and select the **Add New Repository** menu option.
- Enter a name for the new repository in the **Name** field.
- Enter or select the name of a previously configured database connection in the **Database Location** field. The database connection can be added or edited by clicking the **Add** and **Edit** buttons respectively. Database connections can also be added globally by right-clicking on the "Database Connections" tree item in the SOA Security Gateway Management Console and selecting the **Add a new Database Repository** menu option. Take a look at the Database Connection guide for more information on configuring global database connections.

- The **Database Query** is used to retrieve a specific user's profile from the database so that the SOA Security Gateway can then authenticate them. Having successfully authenticated the user, it is possible to select an attribute of this user to use for an authorization filter later on in the policy. The **Database Query** can take the form of a SQL statement, stored procedure, or function call. For more information on how to configure the **Database Query**, please refer to the Database Query Configuration guide.

- If the user sends up a clear-text password to the SOA Security Gateway, but that user's password is stored in a hashed format in the database, it is necessary for the SOA Security Gateway to hash the password before performing the authentication step. Select the **Hash client password** radio button if you want the SOA Security Gateway to do this. Otherwise, select the **Do not hash client password** option.

- If you have selected to hash the client's password, the SOA Security Gateway needs to know the format of the hashed password. The most typical formats are available from the dropdown, however, it is also possible to enter another format. Formats should be entered in terms of message attributes. The following 2 formats are available from the **Hash Format** dropdown. The first option combines the username, authentication realm, and password respectively. This combination is then hashed. The second option simply creates a hash of the user's password.

    - ${authentication.subject.id}:${authentication.subject.realm}:${authentication.subject.password}

    - ${authentication.subject.password}

- Select either **MD5** or **SHA1** from the **Hash Algorithm** radio buttons. This algorithm will be used to create the hash.

- The **Query Result Processing** section of the screen allows you to provide the SOA Security Gateway with some meta information about the result returned by the **Database Query** configured earlier on this screen. In the **Password Column** field, specify the name of the database table column that contains the user's password. The contents of this column will be compared to the password submitted by the user.

- Depending on how the user's password has been stored in the database, select either "Clear Password" or "Digest Password" from the **Password Type** dropdown.

- By running the **Database Query**, all of the user's attributes are returned. Only the user's username and password are used for the actual authentication event. It is also possible to use one of the other user's attributes for authorization at a later stage in the configured policy. The additional "authorization attribute" should be either a username or an X.509 distinguished name (DName). The name of the column containing either the username or the DName should be entered in the **Authorization Attribute Column**, but only if this value is required for authorization purposes.

- The SOA Security Gateway's authorization filters all operate on the basis of a username or DName. In other words, they all evaluate whether a user identified by a username or DName is allowed to access a specific resource. Select the appropriate format from the **Authorization Attribute Format** dropdown depending on what type of user credential is stored in the database table column entered above.

LDAP Repositories:

In cases where an organization maintains user profiles in an LDAP directory, it does not make sense to re-key those profiles into the default local repository. Rather, the SOA Security Gateway can leverage an existing LDAP directory by querying it for user profile data. If a user's profile can be retrieved and successfully validated, that user will be authenticated. Complete the following steps to connect to an LDAP-based authentication repository:

- Right-click on the "LDAP Repositories" tree item in the SOA Security Gateway Management Console and select the **Add a New Repository** menu option.
- Enter a name for the repository in the **Repository Name** field.
- Enter or select a repository from the **LDAP Directory** dropdown. A repository can be added or edited by clicking on the **Add/Edit** button. For more information on configuring the fields on the **Configure LDAP Server** dialog, please refer to the LDAP Configuration guide.
- The **User Search Conditions** section instructs the SOA Security Gateway to search the LDAP tree according to certain conditions. The value entered in the **Base Criteria** field tells the SOA Security Gateway where it should begin searching the LDAP directory from.
- In the **User Class** dropdown, enter or select the name given by the particular LDAP directory to the *User* class. By default, the "User" and "inetOrgPerson" LDAP classes are available for selection, however, it is also possible to add other classes here if necessary.
- When a user is stored in an LDAP directory, a number of user *attributes* are stored along with that user. One of these attributes will correspond to the user name presented by the client for authentication. However, different LDAP directories use different names for that user attribute. For convenience, 3 common attribute names are provided in the **User Search Attribute** dropdown: "cn", "sn", and "uid".
- Check the **Allow Blank Passwords** checkbox to allow the use of blank passwords.
- In an LDAP directory tree, there must be one user attribute that uniquely distinguishes any one user from all the others. This is usually the Distinguished Name of

the user, however this is called different things in different LDAP directories. Two common names are provided by default in the **Login Authentication Attribute** dropdown: "Distinguished Name" and "Entry Domain Name". Select or enter the appropriate option in the dropdown.

- Once the client has been successfully authenticated, it is possible to use any one of that user's stored attributes in a subsequent Authorization filter. In this case we simply want to use the user's Distinguished Name for an Authorization Filter, so we enter "entrydn" into the dropdown. However, any user attribute can be entered in the **Authorization Attribute** field, as long as the subsequent Authorization Filter supports it.
- Since any user attribute can be specified in the **Authorization Attribute** above, it is necessary to inform the SOA Security Gateway of the type of this attribute. This information will be used internally by the SOA Security Gateway in subsequent Authorization filters. Simply select "X.509 Distinguished Name" from the dropdown. For "non-standard" user attributes, "Other" should be selected from this dropdown.

When a filter is configured to *authenticate* a user against an LDAP repository using a username and password combination, the following steps occur:

- A pooled LDAP connection with details corresponding to the repository selected in the **LDAP Directory** field is retrieved.
- A search filter, e.g. (&(objectClass={User})(sAMAccountName={c05vc})), is run using the retrieved connection. Attributes configured in the **Login Authentication Attribute** and **Authorization Attribute** fields are also retrieved in the same search operation.

  For example, if "Distinguished Name" is selected from the dropdown, the user's DName will be retrieved from the LDAP directory. The user's DName uniquely identifies the user within the LDAP directory and is used to bind to the directory so that the user's password can be verified. The attribute specified in the **Login Authentication Attribute** field is used for the bind operation when we bind as any user. The **Authorization Attribute** value will be set to the `authentication.subject.id` and may be used by other filters in the circuit, e.g. for authorization purposes.
- If no results are returned from the search, the user has not been found in the directory. It is important that the administrator user configured on the **Configure LDAP Server** screen has the ability to see the user you are attempting to authenticate.
- If multiple results (i.e. users) are returned from the search, an attempt is made to bind to the directory using each **Login Authentication Attribute** value retrieved from the search, together with the password from the message.

- If more than 1 user is authenticated correctly, an error is returned since we only want to authenticate a single user.
- If no user is authenticated, an error is returned.
- If a single user's **Login Authentication Attribute** value and password binds successfully to the directory, authentication has succeeded.
- Any successful bind is immediately closed.

Database Connections

CA SOA Security Gateway typically connects to databases in order to authenticate or authorize users using the SOA Security Gateway's numerous Authentication and Authorization filters. Similarly, the SOA Security Gateway can retrieve user attributes from a database, which can then be used, for example, to generate SAML attribute assertions at a later stage in the policy.

Individual database connections are configured globally within the SOA Security Gateway Management Console, making them available throughout the various filters that require a database connection. This means that an administrator can simply re-use the same database connection details across multiple authentication, authorization, and attribute-based filters.

The SOA Security Gateway maintains a JDBC pool of database connections to avoid the overhead of setting up and tearing down connections to service simultaneous requests. This pool is implemented by the *Jakarta DBCP (Database Connection Pools)*. The settings in the **Advanced** section of the **Configure Database Connection** dialog (see below) are used internally by the SOA Security Gateway to initialize the connection pool. The table at the end of this section shows how the fields correspond to specific configuration DBCP settings.

To configure details for a global database connection, right-click on the "Database Connections" tree item, which can be found beneath the top-level "External Connections" node on the tree view in the SOA Security Gateway Management Console. Select the **Add a Database Connection** menu option and configure the following fields on the **Configure Database Connection** dialog:

Name:

Enter a name for the database connection in the **Name** field.

URL:

Enter the fully qualified URL of the location of the database.

User Name:

The username to use to access the database.

Password:

The password for the user specified in the **User Name** field.

Initial Size:

The value entered here determines the initial size of the DBCP pool when it is first created.

Maximum Number of Active Connections:

The maximum number of active connections that can be allocated from the JDBC pool at the same time. The default maximum is 8 active connections.

Maximum Number of Idle Connections:

The maximum number of active connections that can remain idle in the pool without extra ones being released. The default maximum is 8 connections.

Minimum Number of Idle Connections:

The minimum number of active connections that can remain idle in the pool without extra ones being created. The default minimum is 8 connections.

Maximum Wait Time:

The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. or -1 to wait indefinitely. The default time is 10000ms and a value of -1 indicates an indefinite time to wait.

Time Between Eviction:

The number of milliseconds to sleep between runs of the thread that evicts unused connections from the JDBC pool.

Number of Tests:

The number of connection objects to examine from the pool during each run of the evictor thread. The default number of objects is 3.

Minimum Idle Time:

The minimum amount of time an object may sit idle in the pool before it is eligible for eviction by the idle object evictor (if any).

The table below shows the correspondence between the fields on the **Database Configuration** screen and the DBCP configuration properties:

Field Name	***DBCP Configuration Property***
URL	url
User Name	username
Password	password
Initial Size	initialSize
Maximum Number of Active Connections	maxActive
Maximum Number of Idle Connections	maxIdle
Minimum Number of Idle Connections	minIdle
Maximum Wait Time	maxWait
Time Between Eviction	timeBetweenEvictionRunsMillis
Number of Tests	numTestsPerEvictionRun
Minimum Idle Time	minEvictableIdleTimeMillis

LDAP Connections

In the same way that database connections can be configured globally in the SOA Security Gateway Management Console (and then re-used across individual filters), LDAP connections are also managed globally within the SOA Security Gateway Management Console. LDAP connections are used by authentication, authorization, and attribute filters. Filters that require the use of a public key (from a public-private key pair) can also retrieve the key from an LDAP source.

When a filter that uses an LDAP directory is run for the first time, it will bind to the LDAP directory using the connection details configured on the **Configure LDAP Server** dialog. Usually the connection details include the username and password of an administrator user who has read access to all users in the LDAP directory for whom we wish to retrieve attributes or authenticate.

To configure a global LDAP connection, right-click on the "LDAP Connection" tree item, which can be found beneath the "External Connections" tree node in the SOA Security Gateway Management Console. Select the **Add an LDAP Connection** menu option and

configure the following fields on the **Configure LDAP Server** dialog:

Name:

Enter a name for the LDAP connection in the **Name** field.

URL:

Enter the location of the LDAP directory in the **URL** field. An LDAP URL is a combination of the protocol (i.e. LDAP), the IP address or host name of the LDAP server, and the port number for the LDAP service. By default, port 389 is reserved for LDAP connections. The following is an example of a typical LDAP directory URL:

ldap://192.168.0.45:389

Authentication Type:

If the configured LDAP directory requires clients to authenticate to it, you must select the appropriate authentication method in the **Type** dropdown. When the SOA Security Gateway connects to the LDAP directory, it will be authenticate to it using the method selected here. The SOA Security Gateway can authenticate to an LDAP directory using the following methods

1. No Authentication:

No authentication credentials need be submitted to the LDAP server for this method. In other words, the client connects anonymously to the server. Typically a client is only allowed to perform "read" operations when connected anonymously to the LDAP server. It is not necessary to enter any authentication details for this authentication method.

2. Simple Authentication:

Simple authentication involves sending a user name and corresponding password in clear-text to the LDAP server. Since the password is passed in clear-text, it is recommended to connect over an encrypted channel, for example, over SSL.

It is not necessary to specify a **Realm** for the when using the "Simple" authentication method. The realm is only used when a hash of the password is supplied (i.e. for Digest-MD5). However, in cases where the LDAP server contains multiple realms, and the specified user name is present in more than one of these realms, then it is at the discretion of the specific LDAP server as to which user name will actually *bind* to it.

Click the **SSL Enabled** checkbox to force the SOA Security Gateway to connect to the LDAP server over SSL. In order to successfully establish SSL connections with the LDAP server, its certificate must be imported into the SOA Security Gateway's certificate store. This can be done using the global Certificates screen.

3. Digest-MD5 Authentication:

With Digest-MD5 authentication, the server generates some data and sends it to the client. The client encrypts this data with its password according to the MD5 algorithm. The LDAP server then uses the client's stored password to decrypt the data and hence authenticate the user.

The **Realm** field is optional here, but may be necessary in cases where the LDAP server contains multiple realms. If a realm is specified here, the LDAP server will attempt to authenticate the user for the specified realm only.

Test Connection:

Once you have entered all the necessary connection details, you can test the configuration by clicking on the **Test Connection** button. This allows you to detect configuration errors at design time, rather than at run-time.

OCSP Connections

The CA SOA Security Gateway can use OCSP (Online Certificate Status Protocol) to validate a certificate against an OCSP responder or group of responders. OCSP requests for certificate validation can be signed by a key from the **Certificate Store** and the response from the OCSP responder can be optionally validated.

An OCSP Connection typically comprises a *group* of OCSP Responder URLs, together with options to sign OCSP requests and validate OCSP responses. To configure a global OCSP Connection, right-click on the "OCSP Connection" tree item beneath the "External Connection" tree node in the SOA Security Gateway Management Console. Select the **Add an OCSP Connection** option from the menu to display the **Certificate Validation - OCSP** dialog. For information on how to configure this dialog, please refer to the OCSP Certificate Validation help page.

It is important to note that once an OCSP Connection has been added in this manner, it will available for selection in the **OCSP Certificate Validation** filter, which can be found under the **Certificate** category of filters in the SOA Security Gateway Management Console.

XKMS Connections

The SOA Security Gateway can also validate certificates against an XKMS (XML Key Management Service) responder or group of responders. An XKMS Connection consists of a group of XKMS responders to validate certificates against, coupled with the signing key to use for signing requests to each of the responders in the group.

To add a global XKMS Connection, right-click on the "XKMS Connection" node in the tree view of the SOA Security Gateway Management Console and select the **Add an**

XKMS Connection menu option to display the **Certificate Validation - XKMS** dialog. For more information on how to configure the fields on this dialog, please refer to the XKMS Certificate Validation help page.

All global XKMS Connections will be available for selection when configuring the **Certificate Validation - XKMS** filter. This saves the administrator from having to re-key XKMS connection details across multiple filters.

SMTP Servers

SMTP Servers are global configuration items that can be configured here and then referenced by SMTP-related filters. Right-click on the "SMTP Filters" node in the tree and select the **Add an SMTP Server** option from the context menu. The following fields must be configured:

Server Name:

Enter the hostname or IP address of the SMTP server.

Port:

Enter the SMTP port on the server specified in the field above. By default, SMTP servers run on port 25.

User Name:

Enter the user name of a registered user that can access the SMTP server.

Password: Enter the password for this user.

URL Connection Sets

URL Groups are used by the SOA Security Gateway's certificate validation filters to round robin between groups of OCSP or XKMS responders. These global groups can simply be re-used when configuring different validation filters with the SOA Security Gateway Management Console. For this reason, URL Connection Sets are labelled in the tree view according to the type of filters where they are available. For example, URL sets under the "OCSP URL Sets" node will be available in the **OCSP Certificate Validation** filter, while sets under the "XKMS URL Sets" will only be available from the **XKMS Certificate Validation** filter.

At run-time, the SOA Security Gateway can round-robin between the responders in the group to ensure that if one of the responders becomes unavailable, SOA Security Gateway can use one of the other responders in the group.

To add a URL Connection Set for a particular category of filters, right-click on the appropriate tree item beneath the "OCSP URL Sets" node in the tree view of the SOA Security Gateway Management Console. Select the **Add a URL Set** menu option to display the **URL Group** dialog. For more information on how to configure this dialog, take a look at the Configuring URL Groups help page.

Remote Host Settings

Overview

Remote Hosts can be used to tweak the way in which the SOA Security Gateway connects to a particular external server or routing destination. The following list illustrates some typical use cases for Remote Hosts when configured within the SOA Security Gateway:

- You want to force the SOA Security Gateway to send *only* HTTP 1.0 requests to a destination server because you know that this server only supports this version of HTTP.
- There are some inconsistencies in the way the destination server supports HTTP and you want to resolve these issues.
- You do not have, or do not wish to use an existing DNS server to resolve hostnames to IP addresses. In this case the **Remote Host** can be used to map a hostname to a specific IP address or addresses.
- You want to explicitly set the timeout, session cache size, input/output buffer size, and other connection-specific settings for a particular destination server. For example, if the destination server is known to be particularly slow, it is possible to set a longer connection timeout for this server.

Remote Hosts are added ***per-process*** by right-clicking on the Process in the tree view of the SOA Security Gateway Management Console and selecting the **Add Remote Host** menu option. The **Remote Host Settings** configuration screen has 2 tabs, which are described in the next 2 sections.

General Configuration

The following configuration options available on the **General** tab:

Host Name:

The host name or IP address of the Remote Host to connect to. If the host name entered in a **Static Router** filter matches the name entered here, the connection-specific settings configured on the **Remote Host** dialog will be used when connecting to this host. This includes the IP address(es) listed on the **General** tab, which will override the default network DNS server mappings, if configured.

Port:

The TCP port on the Remote Host to connect to.

Maximum Connections:

The maximum number of connections to open to a Remote Host. If the maximum number of connections has already been established, the SOA Security Gateway Process will wait for a connection to drop or become idle before making another request. The default maximum is 128 connections.

Force HTTP 1.0:

In cases where the SOA Security Gateway is routing on to a Remote Host that does not fully support the HTTP 1.1 protocol, certain anomalies may occur during the connection. To account for this, it is possible to force the SOA Security Gateway to use the HTTP 1.0 protocol.

Include Content Length in Request:

This option forces the SOA Security Gateway to include the Content-length HTTP header in all requests to this Remote Host.

Include Content Length in Response:

If the SOA Security Gateway receives a response from this Remote Host that contains a Content-length HTTP header, it will return this length to the client if this option is checked.

Addresses to use instead of DNS lookup:

It is possible to add a list of IP addresses (using the **Add** button) that the SOA Security Gateway will use instead of attempting a DNS lookup on the host name provided. Connection attempts will be made to the listed IP addresses on a round-robin basis. This feature is useful in cases where a DNS server is not available or is unreliable.

So, for example, if a **Static Router** filter is configured to route to "www.webservice.com", it will first check if any **Remote Hosts** have been configured with a **Host Name** entry matching "www.webervice.com". If it can't find a matching host, the **Static Router** filter will use whatever DNS server has been configured for the network on which the SOA Security Gateway is running. If it finds a **Remote Host** with matching **Host Name** it will resolve the hostname to the IP addresses listed here.

Furthermore, it will use all the connection-specific settings configured on the **Remote Host** dialog when routing messages to these IP addresses.

Advanced Configuration

The options available on this screen are used when creating sockets for connecting to the Remote Host. Default values are provided for all fields, and these values should only be modified under advice from the Support Team .

The following advanced configuration options are configured on the **Advanced** tab:

Active Timeout:

When the SOA Security Gateway receives a large HTTP request, it reads the request off the network as it becomes available. If the time between reading successive blocks of data exceeds the **Active Timeout**, the SOA Security Gateway will close the connection.

The idea here is to guard against a Remote Host closing the connection while in the middle of sending data. Imagine the Remote Host's network connection is pulled out of the machine while in the middle of sending data to the SOA Security Gateway. When the SOA Security Gateway has read all the available data off the network, it will wait the **Active Timeout** period of time before closing the connection.

Idle Timeout:

The SOA Security Gateway supports the use of HTTP 1.1 persistent connections. The **Idle Timeout** is the time that SOA Security Gateway will wait after sending a message over a persistent connection to the Remote Host before it closes the connection.

Typically, the Remote Host will tell the SOA Security Gateway that it wants to use a persistent connection. The SOA Security Gateway acknowledges this instruction and decides that it will keep the connection open for a certain amount of time after sending the message to the host. If the connection is not reused by within the **Idle Timeout** period, the SOA Security Gateway will close the connection.

Input Buffer Size:

The maximum amount of memory that will be allocated to each request.

Output Buffer Size:

The maximum amount of memory that will be allocated to each response.

Cache Addresses For:

The amount of time for which to cache addressing information after it has been received from the naming service (e.g. DNS).

SSL Session Cache Size:

This setting determines the size of the SSL session cache for connections to the remote host.

A Note on Configuring Watchdogs

It is possible to configure a HTTP Interface to shut down based on certain *conditions*. One such condition is dependent on the SOA Security Gateway being able to contact a particular back-end Web Service running on a Remote Host. To do this, a **HTTP Watchdog** can be configured for a Remote Host in order to poll the endpoint. If the endpoint cannot be reached, the HTTP Interface will be shut down.

To configure the SOA Security Gateway to shut down a HTTP Interface based on the availability of a Remote Host, you must carry out the following steps:

1. Configure a **HTTP Watchdog** for the Remote Host.
2. Configure a **Requires Endpoint** condition on the HTTP Interface.
3. When configuring this condition, select the Remote Host configured in step 1 above (i.e. the one with the associated Watchdog).

Take a look at the HTTP Watchdog help page for more information on adding a watchdog to a Remote Host. For more information on adding Conditions to a HTTP Interface, please refer to the Configuring Conditions for HTTP Interfaces help page.

Using the CA SDK to Write a Filter

Tutorial Overview

CA exposes several powerful APIs as part of its Software Development Kit (SDK) to allow users to build their own bespoke message filters. Users can leverage CA's pluggable and extensible architecture in order to enhance the message processing capabilities of the CA core processing engine.

The purpose of this tutorial is to walk the reader through a step-by-step example of how to build a custom-built message filter with the SDK, and integrate it into a CA policy. In this tutorial we will build the 2 main aspects of an "Addition" filter: the server runtime component and the SOA Security Gateway Management Console configuration component. We will then integrate these components into a CA policy and demonstrate how the filter adds the values of 2 parameters of a SOAP message together and returns the result to the client. By the end of this tutorial, you should be able to write and test your own message filters by following a similar procedure.

Before going any further, however, it is important to explain exactly what are CA filters and to see exactly how they can be wired together to create message processing policies.

Policies, Filters, and Message Attributes

A policy consists of a network of message filters where each filter is a modular executable unit that performs a specific type of processing on a message. The policy arranges these filters into sequences called paths. The filters then act as decision-making points along these paths, determining which filters are run on the message, and in what order.

The following 4-node *policy* contains a single path with 4 *message filters*. The filter marked as *Start* (i.e. "AuthN: WS-Security Username Token") will be executed first. If this filter runs successfully, the next filter in the path (i.e. "getQuotes Operation Name") will be run, and so on until the last filter (i.e. "Echo Web Service") in the path is executed.

When a HTTP request is received, it is converted into a set of message attributes. Each message attribute represents a specific characteristic of the HTTP request, such as the HTTP headers, HTTP body, and MIME parts, amongst others.

Every Filter declares the message attributes that it *requires*, *generates*, and *consumes* from the *attributes blackboard*. The blackboard contains all the available message attributes. When a filter *generates* message attributes it puts them up on the blackboard so that when another Filter *requires* them it can pull them off the blackboard. If a filter *consumes* a message attribute, it is wiped from the blackboard so that no other filter in

the policy can use it.

The following table summarizes the attributes required and generated by the **Operation Name** resolver, which filters incoming requests based on their SOAP operation and namespace:

Filter Name:	Operation Name
Description:	Filters incoming SOAP requests based on their SOAP operation and namespace.
Required Attributes:	content.body http.request.uri
Generated Attributes:	soap.request.method

For more information on policy, policy building, filters, and attributes, please refer to the following documentation:

- A First Tutorial: Protecting the Sample Web Service
- Message Filter Reference
- Message Attribute Reference

It is important that you read the first tutorial, and at least glance over the 2 references documents listed above to familiarize yourself with the concept of CA policies, filters, and message attributes. At the very least, you should understand the following definitions:

- **Policy:**

  A network of interconnected message filters.

- **Message Filters:**

  An executable unit that performs a specific type of processing on message attributes.

- **Message Attributes:**

Message attributes represent specific characteristics of a message.

CA SDK Overview

The CA Software Development Kit (SDK) comprises 3 Java packages that provide programmatic access to CA policies, message objects, and the Entity Store. The following table describes the 3 packages:

Package	***Description***
com.vordel.circuit	This package is responsible for implementing the core CA policy. It includes the base classes for all message filters and their associated classes.
com.vordel.mime	Includes classes that encapsulate the message as it passes through the CA policy. It also provides programmatic access to HTTP headers, HTTP body, request query string (if present), and MIME parts.
com.vordel.es	These classes provide access to the underlying Entity Store where all configuration data is stored.

Tutorial Prerequisites

You should read A First Tutorial: Protecting the Sample Web Service to make sure you understand the concept of policies and filters before continuing.

The CA SDK requires a JDK 1.5, and is supported for the Windows, Linux, and Solaris packages.

CA SDK Sample Overview

The CA SDK ships with a working example of a message filter, called the `SimpleFilter`, that demonstrates how to use the SDK to build a filter. The filter simply extracts 2 integer parameters from a SOAP message, adds the integers, and returns the result of the addition in a SOAP response to the client.

The remainder of this tutorial documents the steps required to build, integrate, configure, and test the supplied `SimpleFilter` and `SimpleProcessor` classes. The steps are as follows:

Step	***Description***
Step 1: Create TypeDocs	Every filter has an associated XML-based TypeDoc that contains the entity's type definition. It defines the configuration field names for that filter and their corresponding data types.
Step 2: Load TypeDocs	Once the TypeDoc has been created for the filter, it must be registered with the Entity Store.
Step 3: Create Filter Class	Every message filter has an associated Filter class that encapsulates the configuration data for a particular instance of the filter. It also returns the corresponding Processor and SOA Security Gateway Management Console classes.
Step 4: Create Processor Class	The Processor class is the server run-time component that is responsible for processing the message. Every message filter has an associated Processor and Filter class.
Step 5: Create SOA Security Gateway Management Console Classes	All Filters are configured through the SOA Security Gateway Management Console. Every Filter has a configuration wizard that allows the user to set each of the fields defined in the entity that corresponds to that Filter. The Filter can then be added to a policy in order to process messages.
Step 6: Build Classes	Once the classes have been written, they must be built and added to the server's classpath.
Step 7: Construct a Policy	We will construct a policy that echoes messages back to the client, and then add the newly created filter to it.
Step 8: Configure the SimpleFilter	We will use the GUI component of the newly added filter to configure its configuration fields. We will then test the functionality of the filter (and its configuration) using the CA **SOAPbox** testing utility.

Step 1: Create the Typedocs

All configuration data is stored as entities in the CA Entity Store. The CA Entity Store is an XML-based store that holds all configuration data required to run the CA core processing engine. Each configurable item has an entity type definition. The entity type definition is defined in an XML file known as the TypeDoc.

Entity types are analogous to class definitions in an object-oriented programming language. In the same way that instances of a class can be created in the form of objects, an instance of an entity type can also be created. Therefore it is useful to think of the entity type defined in a TypeDoc as a header file, and the entity itself as a class instance. All entities and their entity type definitions are stored in the CA Entity Store.

Every filter requires specific configuration data in order to carry out its processing on the message. For example, the `SimpleFilter`, which extracts the values of 2 elements from a SOAP message and adds them together, must be primed with the names and namespaces of those 2 elements.

Since a filter is a configurable item, it requires a new XML typedoc to be written containing an entity type definition for it. The entity type for a filter, contains a set of configuration parameters and their associated data types and default values.

When an instance of the filter is added to a policy via the SOA Security Gateway Management Console, a corresponding entity instance is created and stored in the CA Entity Store. Whenever the filter instance is invoked, its configuration data will be read from the entity instance in the Entity Store.

The following XML shows how the TypeDoc lists the various fields that form the configuration data for the Filter.

```
<entityStoreData>
  <entityType name="SimpleFilter" extends="Filter">
    <!-- Name of filter class that encapsulates the config data -->
    <constant name="class" type="string"
                value="com.vordel.example.filter.SimpleFilter"/>
    <!-- List of config fields, their types, and their default values -->
    <field ... />
    <field ... />
    <field ... />
  </entityType>
<entityStoreData>
```

There are 3 simple rules that all TypeDocs must follow:

- Extend the Filter type
- Define a constant Filter class
- List the configuration fields for the entity

The following table describes the important elements and attributes from the `Simple-Filter` TypeDoc listed above:

Element	***Attribute***	***Description***
<entityStoreData>		The topmost wrapper element for the entire type definition.
<entityType>		Contains the actual type definition, including all its fields and their types.
<entityType>	**name**	The unique name for this type.
<entityType>	**extends**	Entity definitions are hierarchical and can inherit from other higher level types. All filters **must** extend the "Filter" type.
<constant>		A `<constant>` element is used to represent a read-only immutable property of the type.
<constant>	**name**	This attribute contains the name of the read-only property. In the example above, the named property is "class", indicating that the value of this constant is the Java class that encapsulates the defined type. The name of this class **must** be specified as a <constant>.
<constant>	**type**	Specifies the type of the `value` attribute. In this case, the value is the name

Element	*Attribute*	*Description*
		of a Java class, which is just a string.
<constant>	**value**	Contains the value of the named property, which is the name of the Java class that encapsulates this type, i.e. "com.vordel.example.filter.SimpleFilter".
<field>		Contains the definition of a single configuration field for this filter.
<field>	**name**	The name of the configuration field. We will see later on in this tutorial how this name will be used to get and set this property.
<field>	**type**	Specifies the data type of the named configuration field. Supported types include "string", "boolean", "encrypted", and "integer", amongst others.
<field>	**cardinality**	Stipulates how many times this field can appear in an instance of the entity. For example, a cardinality of 1 means that this field can only occur once within an entity.
<field>	**default**	Gives a default value for the configuration field, if appropriate.

The TypeDoc for the `SimpleFilter` appears as follows:

```
<entityStoreData>
  <entityType name="SimpleFilter" extends="Filter">
```

```
    <!-- Name of Filter class that encapsulates this config entity -->
    <constant name="class" type="string"
                value="com.vordel.example.filter.SimpleFilter"/>

    <!-- List of config params, their types, and their default values -->
    <field name="param1" type="string" cardinality="1" default="a"/>
    <field name="param1Namespace" type="string"
              cardinality="1" default="http://startvbdotnet.com/web/"/>
    <field name="param2" type="string" cardinality="1" default="b"/>
    <field name="param2Namespace" type="string"
              cardinality="1" default="http://startvbdotnet.com/web/"/>
  </entityType>
</entityStoreData>
```

All type and related information for the Filter are contained within the top-level `<entityStoreData>` element. The actual Filter type declaration together with its field definitions and types are child elements of the `<entityType>` element. Each field name is specified in the `name` attribute of the `<field>` element, while the type and default value for the field are given in the `type` and `default` attributes, respectively.

It is also possible to provide internationalized log messages by specifying an `<entity<` block of type `InternationalizationFilter` within the `<entityStoreData>` elements.

Now that you understand how the configuration data for the filter is defined, it is time to register the new filter type with the Entity Store.

Step 2: Load TypeDocs

The type definition for the `SimpleFilter` must now be registered with the Entity Store using the **Entity Explorer**. The **Entity Explorer** tool can be used for browsing the entity types and entity instances that have been registered with the Entity Store, as well as adding new entity types.

Once the entity type has been registered, it is guaranteed that any time the server needs to create an instance of the `SimpleFilter`, the instance will contain the correct fields with the appropriate types.

Start the Entity Explorer from the `/bin` directory of your installation using the `esexplorer` startup script. The Entity Explorer appears as follows:

There are 2 main tabs on the Entity Store's interface: **Entities** and **Types**. The **Types** tab lists all currently defined entity types that have been registered with the Entity Store, while the **Entities** tab lists all the instances of entities (e.g. configured filters)

that have been committed to the Entity Store.

It is first necessary to point the Entity Explorer at the management interface exposed by a running instance of the SOA Security Gateway. To do this, right-click on the "Entity Stores" node in the **Entity Hierarchy** tree:

The **Connect to an Entity Store** dialog is displayed as follows:

As stated earlier, the SOA Security Gateway exposes a management service that interfaces to the underlying Entity Store. This is the preferred method of managing the Entity Store. We will now start the SOA Security Gateway (which will connect to the Entity Store) and then point the Entity Explorer at the management service exposed by the SOA Security Gateway. Start the SOA Security Gateway from the `/bin` directory of your product installation.

We will now configure the Entity Explorer to talk to the management service exposed by the SOA Security Gateway. By default this service is available at the following URL, where "HOST" refers to the host name or IP address of the machine on which the SOA Security Gateway is running:

`http://HOST:8090/configuration/policies`.

Enter this address in the **URL** field of the **Connect to an Entity Store** dialog. If you have not already changed the default username and password for the entity store, use the default username "admin" with password "changeme". Otherwise, specify the alternative username and password in the fields provided. Click the **OK** button to connect to the management service on the SOA Security Gateway. A log message should be displayed in the Log panel (at the bottom right-hand corner of the GUI) to confirm that you are connected to the SOA Security Gateway at the specified URL.

Expand the Entity Store filename, and then expand the "System Components" node to display the list of entity instances stored in the Entity Store:

Click on the **Types** tab, expand the node representing the SOA Security Gateway's management interface, e.g. "http://my_host:8090/configuration/policies", and then expand the "Entities (Abstract)" node to display the list of registered entity types. Every one of these entity types will have a corresponding type definition. For example, expand the "Filters" node and then click on the "StaticRouterFilter" entity type in the tree. The names and data types of the fields for this entity type can be seen under the **Details** tab on the right-hand side of the Entity Explorer. To view the type definition for this entity, click the **XML** tab.

We will now add the `SimpleFilter` type definition to the Entity Store. Open the **Entities** tree view, and right-click on the node representing the SOA Security Gateway's management interface. Select the **Load a Type Set** option from the menu.

The **Load a TypeSet** dialog is displayed as follows:

Browse to the location of the `sampleTypeSet.xml` file, which is located in the `/filter` directory of your CA SDK installation. A *TypeSet* file is used to group together one or more TypeDocs so that multiple TypeDocs can be added to the Entity Store in batch mode. The `sampleTypeSet.xml` file appears as follows:

```
<typeSet>
    <!-- SimpleFilter TypeDoc -->
   <typedoc file="SimpleFilter.xml"/>
</typeSet>
```

Important Note:

Do not check the **Reset** checkbox on this dialog, as it purges all configuration data from the Entity Store.

Select **OK** to add the types to the Entity Store. After adding the `SimpleFilter` type, we can now browse to its definition. Open the **Types** tab once again, expand the "Entities (Abstract)" node, and then the "Filters" node. Remember that the `SimpleFilter` type extended the basic `Filter` type. Click on the "SimpleFilter" item in the tree view to view the type's fields and their types, as defined earlier in the type definition.

The Entity Store is now aware of the `SimpleFilter` type. We are now in a position to create an instance of a Filter class that will encapsulate the fields defined in the `SimpleFilter` type.

Step 3: Create Filter Class

A filter class encapsulates the type information defined in an entity's type definition. As such there are class members that correspond to each of the fields in the type definition. At run-time, when the filter is invoked, the filter class is instantiated with the configuration data for the appropriate entity instance. The filter class is responsible for 4 tasks:

1. Store member variables corresponding to fields in the type definition.

2. Specify the message attributes it requires, consumes, and generates.

3. Return the corresponding server runtime class, i.e. the Processor.

4. Return the corresponding SOA Security Gateway Management Console class.

The following code shows the members and methods of the `SimpleFilter.java` example:

```
package com.vordel.example.filter;

import com.vordel.circuit.DefaultFilter;
import com.vordel.circuit.FilterConfigureContext;
import com.vordel.circuit.MessageProperties;
import com.vordel.es.EntityStoreException;

 /**
 The SimpleFilter will contain the local name of the two parameters
 (i.e. a and b) and will also contain the namespace that these
 elements belong to (i.e. http://startvbdotnet.com/web/)
  **/

public class SimpleFilter extends DefaultFilter {

    // the element name of the first parameter
    String param1;
    // the namespace of the first element
    String param1Namespace;
    // the element name of the second parameter
    String param2;
    // the namespace of the second parameter
    String param2Namespace;

    /**
     * Set the message attributes that are used by this filter
     */
    protected final void setDefaultProperties() {
        requiredProperties.add(MessageProperties.CONTENT_BODY);
    }

    /**
     * This method is called to set the config fields for the filter
     * @param ctx The configuration context for this filter
     * @param entity The entity object
     */
    public void configure(FilterConfigureContext ctx,
                          com.vordel.es.Entity entity)
                          throws EntityStoreException {

        super.configure(ctx, entity);

        // read the settings for the processor
        param1  = entity.getStringValue("param1");
        param1Namespace  = entity.getStringValue("param1Namespace");
        param2  = entity.getStringValue("param2");
        param2Namespace  = entity.getStringValue("param2Namespace");
    }

    /**
```

```
     * Returns the server run-time Processor class associated
     * with this Filter class.
     */
    public Class getMessageProcessorClass() {
        return SimpleProcessor.class;
    }

    /**
     * Returns the GUI component for this Filter
     */
    public Class getConfigPanelClass() throws ClassNotFoundException {
        // Avoid any compile or runtime dependencies on SWT and other UI
        // libraries by lazily loading the class when required.
        return
            Class.forName("com.vordel.example.filter.simple.SimpleFilterUI");
    }
}
```

At this stage, it is worth re-visiting the entity definition for the `SimpleFilter` entity to see how the class members correlate to the defined fields.

```
<entityType name="SimpleFilter" extends="Filter">

  <!-- Name of Filter class that encapsulates this config entity -->
  <constant name="class" type="string"
              value="com.vordel.example.filter.SimpleFilter"/>

  <!-- List of config params, their types, and their default values -->
  <field name="param1" type="string" cardinality="1" default="a"/>
  <field name="param1Namespace" type="string"
            cardinality="1" default="http://startvbdotnet.com/web/"/>
  <field name="param2" type="string" cardinality="1" default="b"/>
  <field name="param2Namespace" type="string"
            cardinality="1" default="http://startvbdotnet.com/web/"/>
</entityType>
```

It is important to understand that the Filter class members (i.e. `param1`, `param1Namespace`, `param2`, and `param2Namespace`) directly correspond to the field definitions in the type definition.

These members are populated in the `configure` method of the Filter class, which is called by the framework when the server is started up initially and also whenever the server is refreshed. The `Entity` class provides getter and setter methods for the different data types (e.g. String, boolean, integer, etc). Please refer to the Entity Javadoc for more information.

There are 2 more important methods implemented in this class: `setDefaultProperties` and `getMessageProcessorClass`. The `setDefaultProperties` method allows the Filter to define the message attributes that it *requires*, *generates*, and *consumes* from the attributes blackboard. The blackboard contains all the available message attributes. When a filter *generates* message attributes it puts them up on the blackboard so that when another Filter *requires* them it can pull them off the blackboard. If a filter *consumes* a message attribute, it is wiped from the blackboard so that no other filter in the policy can use it.

The attributes are stored in String arrays (i.e. `reqProps`, `genProps`, `conProps`), which are inherited from the `VariablePropertiesFilter` class. In the case of the `SimpleFilter` class, the `content.body` attribute is required because the SOAP parameters must be extracted from the body of the HTTP request.

The next important method here is the `getMessageProcessorClass` method, which returns the server runtime component (i.e. the Processor class) that is associated with this Filter class. Every Filter class has a corresponding Processor class, which is responsible for using the configuration data stored in the Filter class to process the message. In the next step, we will look at the `SimpleProcessor` class to see how it acts on the data stored in the `SimpleFilter` class.

Finally, the corresponding SOA Security Gateway Management Console configuration class is returned by the `getConfigPanelClass` method, which in this case is the `com.vordel.example.filter.simple.SimpleFilterUI` class. We will look at this class in Step 5 of this tutorial.

Step 4: Create Processor Class

The Processor class is responsible for performing the processing on the message. It uses the configuration data stored in the Filter class to determine how to process the message. It is important to note that this is the server runtime component of the filter that is returned by the `getMessageProcessorClass` of the Filter class described in the previous section.

The following skeleton code shows how the Processor *attaches* to the Filter class and uses its data to process the message. Please note that the following code is for illustration purposes only, and some of the `SimpleProcessor` code has been omitted.

```
package com.vordel.example.filter;

import java.io.ByteArrayInputStream;
import java.io.IOException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
```

```
import com.vordel.circuit.Circuit;
import com.vordel.circuit.CircuitAbortException;
import com.vordel.circuit.Filter;
import com.vordel.circuit.Message;
import com.vordel.circuit.MessageProcessor;
import com.vordel.circuit.MessageProperties;
import com.vordel.es.EntityStore;
import com.vordel.mime.Body;
import com.vordel.mime.ContentType;
import com.vordel.mime.HeaderSet;
import com.vordel.mime.XMLBody;
import com.vordel.trace.Trace;

/**
 * This Processor acts as a simple "Addition" Web Service.
 * It extracts 2 parameters from a SOAP message and adds then together.
 * The result is then returned to the client.
 * The incoming message is expected to be in the following format:

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Add xmlns="http://startvbdotnet.com/web/">
      <a>1</a>
      <b>1</b>
    </Add>
  </soap:Body>
</soap:Envelope>

 The SimpleFilter will contain the local name of the two parameters,
 i.e. a and b, and will also contain the namespace that these elements
 belong to, i.e. http://startvbdotnet.com/web/.
 */
public class SimpleProcessor extends MessageProcessor {

     /**
     * This method attaches the Filter to the Processor object.
     * This is called at startup and on every refresh.
     * This should contain server-side config/initialisation.
     * For example, if this filter is required to establish
     * connections to any 3rd party products/servers, the
     * connection setup should be done here.
     * @param ctx Configuration context for the filter.
     * @param entity The Entity object
     */
    public void filterAttached(FilterConfigureContext ctx,
                               com.vordel.es.Entity entity)
                               throws EntityStoreException {
        // nothing to do here for initialisation
        super.filterAttached(ctx, entity);
    }

    /**
     * The invoke method performs the filter processing.
     * @param c The circuit
     * @param message The message
     * @return true or false.
     */
    public boolean invoke(Circuit c, Message message)
      throws CircuitAbortException {

        try {
            // Get the incoming request message as a DOM
            Document doc = getDOM(message);
```

```
            // Default result
            String result = "UNKNOWN";

            // Cast the filter member variable to a SimpleFilter so that
            // we may access the values stored in the SimpleFilter's
            // fields, e.g. param1, param1Namespace, etc.
            SimpleFilter f = (SimpleFilter)filter;

            // Look into the DOM to get the two parameters.
            // Get the 1st parameter
            NodeList param1 =
                doc.getElementsByTagNameNS(f.param1Namespace, f.param1);
            if (param1 == null || param1.getLength() <= 0)
                throw new CircuitAbortException(
                    "Could not find " + f.param1 + "in message");
            // Get the value passed in the 1st parameter
            String a = getElementContent((Element)param1.item(0));

            // Get the 2nd parameter
            NodeList param2 =
                doc.getElementsByTagNameNS(f.param2Namespace, f.param2);
            if (param2 == null || param2.getLength() <= 0)
                throw new CircuitAbortException(
                    "Could not find " + f.param2 + "in message");

            // Get the value of the 2nd parameter
            String b = getElementContent((Element)param2.item(0));

            // Calculate the result by adding the two parameter values
            result =
                Integer.toString(Integer.parseInt(a) + Integer.parseInt(b));

            // Set the response by setting the content body
            // to be the response
            HeaderSet responseHeaders = new HeaderSet();
            responseHeaders.putString("Content-Type", "text/xml");
            StringBuffer response = new StringBuffer(RESPONSE_START);
            response.append(result);
            response.append(RESPONSE_END);
            Body convertedBody =
                Body.create(responseHeaders, new ContentType("text/xml"),
                    new ByteArrayInputStream(
                        response.toString().getBytes()));
            message.setProperty(
                MessageProperties.CONTENT_BODY, convertedBody);

            return true;
        }
        catch (IOException exp) {
            Trace.error("IOException in SimpleProcessor: " + exp.getMessage());
            return false;
        }
    }
}
```

There are 2 important methods that ***must*** be implemented by every Processor: the `filterAttached` method and the `invoke` method. The `filterAttached` method asso-

ciates an appropriate Filter class with the Processor. The Processor can then access the configuration data stored in the Filter class. In this case, the `SimpleProcessor` attaches to the `SimpleFilter` class. The `filterAttached` method should contain any server-side initialization or configuration that is to be performed by the Filter, such as connecting to 3rd party products or servers.

The `invoke` method is responsible for using the data stored in the attached Filter class to perform the actual message processing. It is this method that is called by the server as it executes the series of filters in any given policy. In the case of the `SimpleFilter`, the `invoke` method extracts the values of the `<a>` and `<b>` elements from the SOAP message, and adds the 2 values together. The result is then returned to the client in a templated SOAP response.

It is important to note that the `invoke` method can have 3 possible outcomes:

Outcome	***Meaning***
True	If the filter processed the message successfully (e.g. successful authentication, schema validation passed, etc), the `invoke` method should return a true result, meaning that the next filter on the success path for the filter will be invoked.
False	If the filter's processing fails (e.g. user was not authenticated, message failed integrity check, etc), the `invoke` method should return false, meaning that the next filter on the failure path for the filter will be invoked.
CircuitAbortException	If for some reason the filter cannot process the message at all - for example if it can't connect to an Identity Management server to authenticate a user - it should throw a CircuitAbortException. If a CircuitAbortException is thrown within a policy, the designated Fault Processor (if any) will be invoked instead of any successive filters on either the success or failure paths.

Now that we have a class to encapsulate the configuration data and another class to act on that data, it is now time to create some GUI classes where the user can configure the fields stored in the Filter class.

Step 5: Create SOA Security Gateway Management Console

Classes

The next step involves writing 2 GUI classes so that the fields defined in the `SimpleFilter` type definition can be configured. Once the GUI classes and resources have been built, the visual components can be used in the SOA Security Gateway Management Console to configure the Filter and add it to a policy.

The following table describes the GUI classes and resources that relate to the `SimpleFilter`:

Class or Resource	***Description***
SimpleFilterUI.java	This class is used to list the **pages** that are involved in a Filter's configuration screen. Each Filter has at least 2 pages: the main configuration page, and a page where log messages related to the filter can be customized. This class is returned by the `getConfigPanelClass` method of the `SimpleFilter` class.
SimpleFilterPage.java	This class defines the layout of the visual fields on the Filter's main configuration screen. For example, there will be 4 text fields on the configuration screen for the `SimpleFilter` corresponding to the 4 fields defined in the entity's type definition.
resources.properties	This file contains all text that appears in the GUI configuration screen, for example, dialog titles, field names, and error messages. This means that the text can be customized or internationalized easily without the need for code change.
simple.gif	This image file is used as the icon to identify the Filter in the Management Console, and will appear in the Filter Palette.

We will first look at the `SimpleFilterUI` class, which is returned by the `getConfigPanelClass` method of the `SimpleFilter` class. It is responsible for the following:

- Listing the configuration pages that make up the interface for the filter
- Naming the category of filters to which this filter belongs

- Specifying the name of the images to use as the icons/images for this filter

The code for the `SimpleFilterUI` is as follows:

```
package com.vordel.example.filter.simple;

import java.util.Vector;

import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.swt.graphics.Image;

import com.vordel.client.manager.Images;
import com.vordel.client.manager.filter.DefaultGUIFilter;
import com.vordel.client.manager.wizard.VordelPage;

/**
 * Filter configuration GUI for 'Simple' example filter.
 * This class shows how to code simple text fields for configuring a
 * custom filter.
 */
public class SimpleFilterUI
    extends DefaultGUIFilter
{
    /**
     * Add the pages that makeup the overall configuation for the Filter.
     */
    public Vector getPropertyPages() {
        Vector<VordelPage> pages = new Vector<VordelPage>();

        // Add the panel for configuring the specific fields
        pages.add(new SimpleFilterPage());

        // Add the page that allows the user to set the log messages
        // for the audit trail, for the pass/fail/error cases
        pages.add(createLogPage());

        return pages;
    }

    /**
     * Set the categories in which we want to display this Filter.
     * The categories define the sections of the palette in which
     * the Filter will appear. The values returned should be the
     * localized name of the palette section, so ensure that the
     * property is defined in the resources.properties in this
     * class' package.  We will add this file to the "Example
     * Filters" category.
     */
    public String[] getCategories() {
        return new String[]{_("FILTER_GROUP_EXAMPLE")};
    }

    // Register our custom images with the image registry
    private static final String IMAGE_KEY = "simpleFilter";
    static {
        Images.getImageRegistry().put(IMAGE_KEY,
                Images.createDescriptor(SimpleFilterUI.class, "simple.gif"));
    }

    /**
```

```
     * Implement this method if you want to display a non-default image
     * for your filter in the policy editor canvas and navigation tree.
     */
    public Image getSmallImage() {
        return Images.get(IMAGE_KEY);
    }

    /**
     * Implement this method to display a non-default icon
     * for your filter in the palette.
     */
    public ImageDescriptor getSmallIcon() {
        return Images.getImageDescriptor(IMAGE_KEY);
    }
}
```

The following table describes the important methods:

Method	***Description***
public Vector getPropertyPages()	Initializes a Vector of the Pages that makeup the total configuration screens for this Filter. Successive Pages are accessible by clicking the **Next** button on the SOA Security Gateway Management Console configuration screen.
public String[] getCategories()	This method returns the names of the Filter categories that this Filter belongs to. The Filter will appear under these categories in the Filter Palette in the SOA Security Gateway Management Console. The `SimpleFilter` will be added to the "Example Filters" category.
public Image getSmallImage()	The default image for the Filter, which is registered in the static block in the code above, can be overridden by returning a different image here.
public ImageDescriptor getSmallIcon()	The default icon for the Filter can be overridden by returning a different icon here.

A *Page* only represents a single configuration screen in the SOA Security Gateway Management Console. It is possible to chain together several Pages to form a series of con-

figuration screens that together make up the overall configuration for a given Filter. By default, all Filters consist of 2 pages: one for the configuration fields for the Filter, and the other to allow per-Filter logging. However, there is no reason why more Pages can not be chained together. Successive Pages should be added to the configuration in the `getPropertyPages` method.

From looking at the `getPropertyPages` method of the `SimpleFilterUI`, it is clear that the `SimpleFilterPage` class forms one of the configuration screens (or pages) for the `SimpleFilter` filter. The `SimpleFilterPage` class is responsible for the layout of all the input fields that make up the configuration screen for the `SimpleFilter`. The code for the `SimpleFilterPage` class is shown below:

```
package com.vordel.example.filter.simple;

import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;

import com.vordel.client.manager.wizard.VordelPage;

public class SimpleFilterPage extends VordelPage
{
    /**
     * Create the configuration page. Set the title and description
     * here. The title and description are maintained in the
     * resources.properties file for customization and
     * internationalization purposes.
     */
    public SimpleFilterPage() {
        // Call the super constructor with a unique name for this
        // page to bind it with its corresponding wizard.
        super("simplePage");
        setTitle(_("SIMPLE_PAGE"));
        setDescription(_("SIMPLE_PAGE_DESCRIPTION"));
        setPageComplete(false);
    }

    /**
     * Get the unique identifier for the help page for this filter.
     * The id-to-page mapping is maintained in the
     * '/docs/res/help.jhm' file. The URL for the page to describe
     * the filter should be relative to the '/docs' directory.
     */
    public String getHelpID() {
        return "simple.help";
    }

    /**
     * Any post-processing of the configuration values can happen
     * here, before they are persisted to the Entity Store. If the
     * configuration is not complete and valid, notify the user here
     * with a dialog box and return false here, otherwise return true.
     *
     * @see com.vordel.client.manager.util.MsgBox
     */
    public boolean performFinish() {
        // Simple mutually independent values here, no checking required,
        // so return true
```

```
        return true;
    }

    /**
     * Create the main control for the Filter configuration dialog.
     * This will house the text fields for setting the particular
     * Field Values in the Entity.
     */
    public void createControl(Composite parent) {
        // Create a Panel with two columns
        GridLayout layout = new GridLayout();
        layout.numColumns = 2;
        Composite container = new Composite(parent, SWT.NULL);
        container.setLayout(layout);

        // Add controls to populate the appropriate Entity Fields
        // We use the localization keys for the field names and
        // descriptions which will map to entries in the
        // resources.properties file.
        createLabel(container, "SF_NAME");
        createTextAttribute(container, "name", "SF_NAME_DESC");

        createLabel(container, "SF_PARAM1");
        createTextAttribute(container, "param1", "SF_PARAM1_DESC");

        createLabel(container, "SF_PARAM1NS");
        createTextAttribute(container, "param1Namespace", "SF_PARAM1NS_DESC");

        createLabel(container, "SF_PARAM2");
        createTextAttribute(container, "param2", "SF_PARAM2_DESC");

        createLabel(container, "SF_PARAM2NS");
        createTextAttribute(container, "param2Namespace", "SF_PARAM2NS_DESC");

        // Finish up the page definition
        setControl(container);
        setPageComplete(true);
    }
}
```

There are 4 important interface methods that must be implemented in this class:

Method	***Description***
public SimpleFilterPage()	The constructor performs some basic initialization, such as setting a unique ID for the page and setting the title and description for the page. The text representing the page title and description are kept in the `resources.properties` file so that they can be localized or customized easily, if necessary.
public String getHelpID()	This method is called by the SOA Security

Method	***Description***
	Gateway Management Console help system. There is a **Help** button on every configuration page in the SOA Security Gateway Management Console. When this button is pressed, the help system is invoke. Every page has a help ID (e.g. "simple.help") associated with it, which is mapped to a help page in the `/docs/res/help.jhm` file, for example: `<mapID target="simple.help" url="/common/tutorials/general_filter.html"/` All URLs specified in the `help.jhm` file are relative from the `/docs` folder of your product installation. You can add this line to the `help.jhm` file now, and we will check that the help page works at the end of this tutorial.
public boolean performFinish()	This method gives you the chance to process the user-specified data before it is submitted to the Entity Store. For example, any validation on the data should be added to this method.
public void createControl(Composite parent)	This method is responsible for creating and ordering the input fields on the configuration page. Once again, localization keys from the `resource.properties` file are used to give labels for the input fields. It is important to note that the `createTextAttribute` takes a String as its 2nd parameter, which corresponds to a field defined for an entity, e.g. "param1", "param1Namespace", "param2", and "param2Namespace" are all defined in the `SimpleFilter` entity type. When the user submits the values entered in these fields,

Method	***Description***
	the values will be set to the corresponding fields in the entity instance in the Entity Store.

Both the `SimpleFilterUI` and the `SimpleFilterPage` classes use localized keys for all text that appears on the configuration screen. This makes it a trivial task to localize or customize all text that appears in the SOA Security Gateway Management Console. The localization keys and their corresponding strings are stored in the `resources.properties` file, which takes the following format:

```
# Palette category for example filters
FILTER_GROUP_EXAMPLE=Example Filters

# Title and Description for the SimpleFilter
SIMPLE_PAGE=Simple Filter Configuration
SIMPLE_PAGE_DESCRIPTION=Configure parameter values for the Simple Filter

#
# Field labels and descriptions
#
SF_NAME=Filter Name:
SF_NAME_DESC=The name of the Simple Filter
SF_PARAM1=Parameter 1
SF_PARAM1_DESC=the first parameter
SF_PARAM1NS=Parameter 1 Namespace
SF_PARAM1NS_DESC=the first parameter namespace field
SF_PARAM2=Parameter 2
SF_PARAM2_DESC=the second parameter
SF_PARAM2NS=Parameter 2 Namespace
SF_PARAM2NS_DESC=the second parameter namespace field
```

The last resource that we must mention is the `simple.gif` image file, which will appear as the icon for the `SimpleFilter` in the SOA Security Gateway Management Console. We will see this icon a little later in this tutorial when we are actually configuring the `SimpleFilter`.

Now that all classes and resources have been written, it is now time to build the relevant Jar files and incorporate them into the product.

Step 6: Build Classes

The following steps ***must*** be carried out in order to build the Java classes (and resources) described in the previous sections:

- Build classes and associated resources into a Jar file using your chosen build technology.
- Place the new Jar in the `/lib` directory of your product installation.
- Place any 3rd party Jars used by you classes into the `/lib` directory as well.

As an example of one way to build the `SimpleFilter` classes, an Ant build file, called `build.xml`, is supplied in the `/example/filter` directory of the SDK. Instructions on how to build are provided in the `readme.html` file, which also resides in this directory.

The Ant file builds the `SimpleFilter` classes and packages all associated resources into the `VordelExampleFilters.jar` file. This file must then be placed in the `/lib` folder as outlined above.

When both the server and the SOA Security Gateway Management Console boot up, they will automatically pick up the new Jar file from the `/lib` directory. In the remaining steps of this tutorials, we will see how to configure a policy that includes the `SimpleFilter` and then test its functionality.

Step 7: Construct a Policy

This section assumes that you have read the Introduction to Policies and the guide on Policy Building, which are available from the "General Configuration" section of the product documentation. First of all, we will build a simple policy that echoes messages back to the client, and then in the next step, we will add the **SimpleFilter** to the policy.

Policies are built using the **Policy Editor** of the **SOA Security Gateway Management Console**. To build a policy, message filters are dragged and dropped from the **Filters Palette** onto the Editor, and then linked by *success* or *failure paths* to create a network of filters. The following screen shot shows the **Policy Editor**:

The **Policy Editor** is the large blank area on the screen, while the **Filters Palette** is the area to the right of it that contains all the filters. Message filters are grouped together by category so that, for example, all the content-based filters appear together in one group, while all the authentication filters are together in a different group. As stated earlier, policies are built by dragging these filters and dropping them onto the Editor.

It is important to note that if you have followed the steps outlined above, you will see a new category of filters, called "Example Filters" in the Filter Palette. The "Example Filters" category is expanded in the screenshot above.

To create a policy, right-click on **Policies** in the tree view to the left of the Editor, and select **Add Policy**. Enter "Circuit 1" as the name of the new policy in the **Policy** dialog.

We will create a policy containing only 1 filter: the **Reflect** filter. This filter simply echoes the client message back to the client. The **Reflect** filter can be found in the **Utility** filter group. Drag this filter on to the editor.

Simply enter a name for the filter (or use the default) in the field provided. Select the default value (i.e. "200") for the **HTTP response status code** and click the **Finish** button.

We now have to configure the Process to invoke the new policy. This can be done by right clicking on the **Firewall** and selecting the **Edit** button.

Enter the following values on the **Configure Relative Path** dialog:

1. **Relative Path:**

   Enter "/" in this field meaning that the Process will invoke the policy selected below for all requests received on this path.

2. **Policy:**

   Select "Circuit 1" to configure the server to send all requests received on the path configured above to our newly configured policy.

To force the server to pick up the new configuration, we must flush the configuration updates. To do this, right click on the process in the tree view and select the **Refresh Server** menu option.

To test this, start up the **SOAPbox** testing tool from the `/bin` directory of your product installation. The following screenshot shows the **SOAPbox** tool:

Assuming your HTTP interface is listening on port 8080, we will configure **SOAPbox** to send a request to: `http://HOST:8080/`, where HOST is the hostname or IP address of the machine on which the server is running. Note that we will send to "/" because, earlier, we have configured the firewall to filter requests received on this relative path.

Copy any SOAP message into the **Request** panel of **SOAPbox**. Click on the **Send** button to send the message to the server, which will then echo it back to the client using the **Reflect** filter configured earlier. When the message has been returned to

, try changing the message slightly to assure yourself that the correct message is actually being returned.

Finally, it is time to add the `SimpleFilter` to the policy, and to test its functionality.

Step 8: Configure the SimpleFilter

In this final section of the SDK tutorial, we will add the `SimpleFilter` to the policy we built in the previous section. Currently, the policy consists of only 1 filter, the **Reflect** filter:

The **SimpleFilter** can be found in the **Example Filters** category of the **Filter Palette** on the SOA Security Gateway Management Console.

Drag and drop the **SimpleFilter** on to the policy editor. The configuration screen appears as follows:

Since the type definition for the `SimpleFilter` entity contained default values, the input fields on the configuration screen are pre-populated with these default values. Before completing the configuration, make sure the help system is working correctly. Remember that in Step 5 of this tutorial, we added a mapping to the `help.jhm` (in the INSTALL_DIR/docs/res folder, where INSTALL_DIR points to the root of your product installation) between the help ID specified in the `SimpleFilter` class, and to this help page, i.e. `/common/tutorials/general_filter.html`.

If you have already added this line to the `/docs/res/help.jhm` file, try clicking the **Help** button. Otherwise, add the following line now, restart the SOA Security Gateway Management Console, and then click the **Help** button.

```
<mapID target="simple.help" url="/common/tutorials/general_filter.html"/>
```

Simply click on the **Finish** button to complete the configuration.

Right-click on the "Simple" node and select **Set as Start** from the context menu. Connect the "Simple" node to the "Reflect" node with a *success* path. This can be done by clicking on the **Success Path** arrow, and then clicking on the "Simple" node, followed by clicking on the "Reflect" node. The policy now appears as follows:

To force the server to pick up the new configuration, we must flush the configuration

updates. To do this, right click on the process in the tree view and select the **Refresh Server** menu option.

We will now test the configuration to make sure that it performs as we expect, i.e. that it can correctly add the 2 numbers together. Load the appropriate SOAP message into the **SOAPbox** by selecting the **File -> Samples -> 1 + 2** menu option. The following SOAP message will be loaded:

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Add xmlns="http://startvbdotnet.com/web/">
      <a>1</a>
      <b>2</b>
    </Add>
  </soap:Body>
</soap:Envelope>
```

It is important to note the presence of the <a> and <b> elements in the SOAP message, as well as the namespace declaration in the <Add> element. These elements and their corresponding namespaces match the values configured in the **SimpleFilter** earlier.

Make sure to send the message to the same address as before by typing in, `http://localhost:8080/` as the URL. The **SOAPAction** field is not needed and can be removed. Press the **Send** button when you have done this to send the message to the server.

The following response will be returned to **SOAPbox**:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance\"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <AddResponse xmlns="http://startvbdotnet.com/web/">
      <AddResult>3</AddResult>
    </AddResponse>
  </soap:Body>
</soap:Envelope>
```

The value of the <AddResult> element is 3, which clearly indicates that the newly ad-

ded filter has worked successfully.

Conlusion

This tutorial described a working example of how to write a message processing filter using the CA SDK and how to integrate it into a CA policy. The reader is now advised to try to build their own filter by following a similar sequence of steps to those outlined above.

If you have any queries on the content of this document, please contact the Support Team with your questions.

CA SOA Security Manager Authentication

Overview

CA SOA Security Manager can authenticate end-users and authorize them to access protected Web resources. When the SOA Security Gateway receives a message containing user credentials, it can forward the message to CA SOA Security Manager where the passed credentials will be extracted from the message in order to authenticate the end-user. Once the message has been passed to CA SOA Security Manager, it can authenticate the user by the following methods:

- **XML Document Credential Collector:**

  Gathers credentials from the message and maps them to fields within a user directory.

- **XML Digital Signature:**

  Validates the X.509 certificate contained within an XML-Signature on the message.

- **WS-Security:**

  Extracts user credentials from WS-Security tokens contained within the message.

- **SAML Session Ticket:**

  Consumes a SAML session ticket from a HTTP header, SOAP envelope, or session cookie to authenticate the end-user.

By delegating the authentication decision to CA SOA Security Manager, the SOA Security Gateway acts as a Policy Enforcement Point (PEP). It *enforces* the decisions made by the CA SOA Security Manager, which acts a Policy Decision Point (PDP).

Please refer to the Authentication Methods section of the CA SOA Security Manager Policy Configuration Guide for more information on these authentication methods.

Configuration

Configure the following fields:

Name:

Enter a name for the filter in this field.

Agent Name:

In order to act as a PEP for the CA SOA Security Manager, the SOA Security Gateway must have been set up as a *SOA Agent* with the SiteMinder Policy Server. Please refer to the **CA SOA Security Manager Agent Configuration Guide** for more information on how to do this.

Enter the SOA Security Gateway's SOA Agent name in **Agent Name** field. At runtime, the SOA Security Gateway will connect as this agent to a CA SOA Security Manager. To configure the location of the CA SOA Security Manager, click on the **Add** button to display the **CA SOA Security Manager Connection Details** dialog.

Enter the name of the agent in the **Agent Name** field before configuring the Host connection details.

The SOA Security Gateway uses the information in the Security Manager hosts configuration file to connect to CA SOA Security Manager. This file is called "SmHost.conf" by default, and is generated after creating a host with CA SOA Security Manager. Browse to the location of this file using the button at the bottom right-hand corner of the **Connection Details** text area. After selecting the configuration file, the connection details will appear in the text area.

- **IP Address:**

  The address of the machine on which CA SOA Security Manager is running.

- **User Name:**

  The username of a CA SOA Security Manager administrator as configured during the CA SOA Security Manager install.

- **Password:**

  The password for the above user.

- **Name of Host to be Registered:**

  The name of the *Trusted Host* to be added to CA SOA Security Manager.

- **Name of Host Configuration Object:**

  The name of the *Host Configuration Object* to be associated with the new host.

The new host will now be registered with CA SOA Security Manager.

There is one last field on this dialog that must be configured to complete the connection details. The **XMLSDKAcceptSMSessionCookie** setting controls whether or not the CA SOA Security Manager authentication filter will accept a single sign-on token for authentication purposes. The single sign-on token must reside in the HTTP header field named "SMSESSION" in order to authenticate via this mechanism. This token is created and updated when the CA SOA Security Manager authorization filter runs successfully.

When the checkbox is selected, the authentication filter allows authentication via a single sign-on token. Note that if no single sign-on token is present within the message, the authentication filter will authenticate fully by gathering credentials from the request in whatever manner has been configured within the CA SOA Security Manager. When the checkbox is deselected, the authentication filter will authenticate fully, i.e. it will never allow authentication via a single sign-on token.

A Note on the xmltoolkit.properties File

The `xmltoolkit.properties` file contains default properties used by the SOA agent, such as the URL of the CA SOA Manager, an identifier for the SOA agent, and an indication to the SOA Manager if it should perform fine-grained resource identification or not. The following example shows the format of the `xmltoolkit.properties` file, which can be found in the `/lib/modules` directory of your the SOA Security Gateway installation:

```
#Wed Jul 18 15:02:16 BST 2007
WSDMEndPointUrl=http\://10.0.7.233\:8282/wsdm71mmi/services/WSDM71MMI
WSDMCustomHandlerType=101
WSDMResourceIdentification=yes
```

The following properties are available:

- **WSDMEndPointUrl:**

  Specifies the full URL of the SOA Manager.

- **WSDMCustomHandlerType:**

  Identifies the agent or observer, i.e. SOA Security Gateway.

- **WSDMResourceIdentification:**

This value cannot be configured from the SOA Security Gateway Management Console GUI and so can only be set directly in the properties file. If this property is set to "no" (or if the properties file cannot be found) only a "coarse-grained" resource identification will be performed on the requested URL. If this value is set to "yes", a "fine-grained" resource identification including the requested URL, Web Service name, and SOAP operation, i.e. [url]/[web service name]/[soap operation]

Note the following points about the `xmltoolkit.properties` file:

- The properties file is written to the `/lib/modules` directory when a SOA Security Manager Authentication or Authorization filter is loaded at startup or on server refresh, but only if the file does not already exist in this location.
- If the properties file already exists in the `/lib/modules` directory, the **WSDMResourceIdentification** property is *not* overwritten. In other words, the user is allowed to manually set this property independently of the SOA Security Gateway Management Console.
- If the **WSDMResourceIdentification** property does not exist, it is given a default value of "yes" and written to the file.

Please refer to the CA SOA Security Manager documentation for more information.

SSL Authentication

Overview

A client can mutually authenticate to the SOA Security Gateway through the exchange of X.509 certificates. An X.509 certificate contains identity information about its owner and is digitally signed by the Certificate Authority that issued it.

A client will present such a certificate to the SOA Security Gateway while the initial SSL/TLS session is being negotiated, in other words, during the *SSL handshake*. The **SSL Authentication** filter extracts this information from the client certificate and sets it as message attributes. These attributes can then be used by subsequent filters in the policy.

The **SSL Authentication** filter can be used as a decision-making node on the policy. For example, it can be used to determine a path through a policy based on how users authenticate to the SOA Security Gateway.

Configuration

Name:

Enter a name for the filter on the **SSL Authentication** configuration screen.

References

SSL 3.0 Specification [http://wp.netscape.com/eng/ssl3/draft302.txt]

IP Address

Overview

The SOA Security Gateway can be configured to allow or deny machines, or groups of machines, access to Web Services based on IP address. The main table on the screen shows the IP addresses from whom the SOA Security Gateway will accept or deny messages depending on what is configured.

Configuration

The following fields must be configured:

Name:

Enter a name for the filter.

IP Addresses:

IP Addresses can be added by clicking the **Add** button, which displays the **Add IP Filter** dialog. Enter an **IP Address** and **Subnet Mask** to indicate a network to filter.

Messages sent from hosts belonging to this network will be accepted or rejected based on what is configured in the section below. A **Subnet Mask** of 255.255.255.255 can be used to filter specific IP addresses.

Important Note:

It is worth noting that if requests are made across a proxy, portal, or other such intermediary, the SOA Security Gateway will filter on the IP address of the intermediary. Therefore, the IP address of the intermediary should be entered on this screen, and not that of the end user/client machine.

Existing IP addresses can be edited and removed by selecting the **Edit** and **Remove** buttons.

Access:

Depending on which radio button - **Allow Access** or **Deny Access** - is checked, the IP addresses listed in the table will either be allowed or denied access to the Web Service.

HTTP Basic Authentication

Overview

A client can authenticate to the SOA Security Gateway with a username and password combination using *HTTP Basic Authentication*. Whenever a **HTTP Basic Authentication** filter is configured, the SOA Security Gateway will request the client to present a username and password combination as part of the *HTTP Basic challenge-response* mechanism.

With HTTP Basic Authentication, the client's username and password are concatenated, base64-encoded, and passed in the "Authorization" HTTP header as follows:

```
Authorization: Basic dm9yZGVsOnZvcmRlbA==
```

The SOA Security Gateway can then authenticate this user against a user profile stored in the SOA Security Gateway's local repository, a database, or an LDAP directory.

Configuration

The information specified on this screen informs the SOA Security Gateway where it can find user profiles for authentication purposes. The SOA Security Gateway can lookup user profiles in the SOA Security Gateway's local repository, a database, or in an LDAP directory. Users can be added to the local repository using the **Users** interface. Take a look at the Users tutorial for more information on how to do this.

To configure the **HTTP Basic Authentication** filter, complete the following fields.

Name:

Enter a name for the filter here.

Realm:

The **Realm** entered here will be presented to the client at the same time as they are entering their username and password. The client is then said to be logging into this realm. It is useful in cases where a given user might belong to many different realms, and so, by presenting the realm to the client, he can specify which realm he wants to log into.

Credential Format:

The username presented to the SOA Security Gateway during the HTTP Basic handshake can be of many formats, usually either username or distinguished name. Since the SOA Security Gateway has no way of inherently telling one format from the other (i.e. the client's username could be a DName), it is necessary to specify the format of the credential presented by the client. This format is then used internally by the SOA Security Gateway when performing authorization lookups against third party Identity Management servers.

Allow Client Challenge:

HTTP Basic Authentication can be configured to work in two ways:

1. **Direct Authentication:**

   The client sends up the "Authorization" HTTP Basic Authentication header in its first request to the server.

2. **Challenge-Response Handshake**

   The client does *not* send the "Authorization" header when sending its request to the server (i.e. it does not know that the server requires HTTP Basic Authentication). The server responds with a *HTTP 401 response code*, instructing the client to authenticate to the server by sending up the "Authorization" header. The client then sends up a second request, this time including the "Authorization" header and the relevant username and password.

The first case is used mainly for machine-to-machine transactions in which there is no human intervention. The second case is typical of situations where a browser is talking to a Web Server. When the browser receives the HTTP 401 response to its initial request, it pops up a dialog to allow the user to enter the username and password combination.

If you wish to force clients to always send the HTTP Basic "Authorization" header to the SOA Security Gateway, disable the **Allow client challenge** checkbox. If, on the other hand, you wish to allow clients to engage in the HTTP Basic Authentication challenge-response handshake with the SOA Security Gateway, make sure this feature is enabled by checking this option.

Remove HTTP Authentication Header

Select this checkbox to remove the HTTP `Authorization` header from the downstream message. If this option is left unchecked, the incoming `Authorization` header will be

forwarded onwards to the destination Web Service.

Repository Name:

The **Repository Name** field specifies the name of the **Authentication Repository** where all **User** profiles are stored. As stated earlier, this can be in the SOA Security Gateway's local repository, a database, or an LDAP directory.

You can select a pre-configured **Repository Name** from the dropdown, or add a new one by selecting the **Add** button. Existing repositories can be edited and removed by clicking the **Add** and **Remove** buttons. For more information on configuring this, take a look at the Authentication Repository tutorial.

References

HTTP Authentication [http://www.ietf.org/rfc/rfc2617.txt]

XML Signature Authentication

Overview

The SOA Security Gateway can authenticate a client by validating the XML Signature contained within an incoming XML message. A successful signature validation proves that the client had access to the signing key. Since the signing key is only accessible by the client (i.e. is not publicly available), the validation process authenticates the client.

Configuration

The following sections can be configured on the **XML Signature Authentication** screen:

Signature Location

There may be multiple signatures present in a given XML message. For this reason, it is necessary to tell the SOA Security Gateway which signature it should use to authenticate the client.

The signature can be extracted:

- Using WS-Security Actors
- From the SOAP Header
- Using XPath

Select the most appropriate method from the **Signature Location** dropdown. Your selection will depend on the types of SOAP messages that you expect to receive. For example, if incoming SOAP messages will contain an XML Signature within a WS-Security block, you should choose this option from the dropdown.

What Must be Signed

This section defines the content that must be signed in order for a SOAP message to pass the filter. This ensures that the client has signed something meaningful (i.e. part of the SOAP message) as opposed to some arbitrary data that would pass a "blind" signature validation. This further strengthens the authentication process.

An XPath expression is used to identify the nodeset that should be signed. To specify that nodeset, select either an existing XPath expression from the **XPath Expression**

dropdown list, or add a new one using the **Add** button. XPath expressions can also be edited and removed with the **Edit** and **Remove** buttons respectively.

Signer's Public Key/Certificate

Select the **Certificate in Message** radio button in order to use the certificate from the XML-Signature specified in the **Signature Location** section. The certificate will be extracted from the `KeyInfo` block.

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
  ...
  <dsig:KeyInfo>
    <dsig:X509Data>
      <dsig:X509SubjectName>CN=Sample User...</dsig:X509SubjectName>
      <dsig:X509Certificate>
        MIIE ....... EQgJ
      </dsig:X509Certificate>
    </dsig:X509Data>
    <dsig:KeyValue>
      <dsig:RSAKeyValue>
        <dsig:Modulus>
          AMfb2tT53GmMiD
          ...
          NmrNht7iy18=
        </dsig:Modulus>
        <dsig:Exponent>AQAB</dsig:Exponent>
       </dsig:RSAKeyValue>
    </dsig:KeyValue>
  </dsig:KeyInfo>
</dsig:Signature>
```

Clients may not always want to include their public keys in their signatures. In such cases, the public key can be retrieved from a specified LDAP directory or from the **Certificate Store**.

To retrieve a client certificate from an LDAP directory, select a pre-configured one from the **LDAP Source** dropdown, or add/edit a new/existing LDAP directory by clicking the **Add/Edit** button.

Alternatively, select the name of a **User** from the **Certificate** field. This user's certificate will then be associated with the incoming message so that all subsequent certificate-based filters will use this user's certificate.

References

XML Signature [http://www.w3.org/Signature/]

XML Signature Syntax and Processing [http://www.w3.org/TR/xmldsig-core/]

Introduction to Public-Key Cryptography [http://docs.sun.com/source/816-6154-10/]

CA SOA Security Manager Authorization

Overview

CA SOA Security Manager can authenticate end-users and authorize them to access protected Web resources. The SOA Security Gateway can interact directly with CA SOA Security Manager by asking it to make authorization decisions on behalf of end-users that have successfully authenticated to the SOA Security Gateway. CA SOA Security Manager decides whether or not to authorize the user and relays the decision back to the SOA Security Gateway where the decision is enforced. The SOA Security Gateway, therefore, acts as a Policy Enforcement Point (PEP) in this situation, enforcing the authorization decisions made by the CA SOA Security Manager, which acts a Policy Decision Point (PDP).

It is important to note that a CA SOA Security Manager authentication filter must be invoked before a CA SOA Security Manager authorization filter in a given policy. In other words, the end-user must authenticate to CA SOA Security Manager before they can be authorized for a protected resource.

Configuration

Configure the following fields on the **CA SOA Security Manager Authorization** filter:

Name:

Enter a name for the filter here.

Attributes:

If the end-user is successfully authorized, the attributes listed here will be looked up in CA SOA Security Manager and returned to the SOA Security Gateway. These attributes will be stored in the `attributes.lookup.list` message attribute. They can be retrieved at a later stage to generate a SAML attribute assertion.

Check the **Set attributes for SAML Attribute token** checkbox, and then click the **Add** button to specify an attribute to fetch from CA SOA Security Manager.

Attributes

Overview

The purpose of the filters in the **Attributes** filter group is to extract user attributes from various sources. It is possible to retrieve attributes from the message, an LDAP directory, a database, the **User Store**, HTTP headers, and finally, from a SAML attribute assertion.

Having retrieved a set of user attributes, the SOA Security Gateway then stores them in the `attribute.lookup.list` message attribute. It is the role of the **Attributes** authorization filter to check the value of these attributes in order to authorize the user.

Configuration

The following fields are available on the **Attributes** configuration screen:

Name:

Enter a name for this filter here.

Attributes:

The **Attributes** table lists the checks that the SOA Security Gateway will perform on user attributes stored in the `attribute.lookup.list` message attribute. The following points describe how the SOA Security Gateway carries out the checks listed in the table.

- The entries in the table are OR-ed together so that if any one of them succeeds, the filter will return a "pass" result.
- The attribute checks listed in the table will be run in series until one of them passes.
- It is possible to add a number of attribute-value pairs to an attribute check by separating them with commas, e.g. "company=ca, department=engineering, role=engineer".
- If multiple check are present in a given attribute check, individual checks are AND-ed together so that the overall attribute check will only "pass" if all of its checks "pass".

To add an attribute check to the **Attributes** table, click the **Add** button. Attributes can then be entered in the **Add Attributes** dialog.

For attribute checks involving attributes extracted from a SAML attribute assertion, it is necessary to specify the namespace of the attribute as it was given in the assertion. So, for example, the SOA Security Gateway can extract the "role" attribute from the following SAML `<Attribute Statement>` and store it in the `attribute.lookup.list`:

```
<saml:AttributeStatement>
 <saml:Attribute Name="role" NameFormat="http://www.company.com">
  <saml:AttributeValue>admin</saml:AttributeValue>
 </saml:Attribute>
 <saml:Attribute Name="email" NameFormat="http://www.company.com">
  <saml:AttributeValue>joe@company.com</saml:AttributeValue>
 </saml:Attribute>
 <saml:Attribute Name="dept" NameFormat="">
  <saml:AttributeValue>engineering</saml:AttributeValue>
 </saml:Attribute>
</saml:AttributeStatement>
```

The "NameFormat" attribute of the `<Attribute>` gives the namespace of the attribute name. This namespace must be entered (together with a corresponding prefix) in the **Add Attributes** dialog.

For example, to extract the "role" attribute from the SAML attribute statement above, you should enter "pre:role=admin" in the **Attribute Requirement** field. Then you must also map the "pre" prefix to the "http://www.company.com" namespace, as specified by the "NameFormat" attribute in the attribute statement.

Certificate Attributes

Overview

The SOA Security Gateway can authorize access to a Web Service based on the X.509 attributes of an authenticated client's certificate. For example, a simple **Certificate Attributes** filter might only authorize clients whose certificates have a Distinguished Name (DName) containing the following attribute: *O=ca*. In other words, only "ca" users are authorized to access the Web Service.

An X.509 certificate consists of a number of fields. The `Subject` field is the one of most relevance to this tutorial. It gives the DName of the client to which the certificate belongs. A DName is a unique name given to an X.500 directory object. It consists of a number of attribute-value pairs called Relative Distinguished Names (RDNs). Some of the most common RDNs and their explanations are as follows:

- `CN`: CommonName
- `OU`: OrganizationalUnit
- `O`: Organization
- `L`: Locality
- `S`: StateOrProvinceName
- `C`: CountryName

For example, the following is the DName of the *sample.p12* client certificate supplied with the SOA Security Gateway:

```
CN=Sample Cert, OU=R&D, O=Company Ltd., L=Dublin 4, S=Dublin, C=IE
```

Using the **Certificate Attributes** filter, it is possible to authorize clients based on, for example, the "CN", "OU", or "C" in the DName.

Configuration

The **X.509 Attributes** table lists a number of attribute checks that will be run against the client certificate. Each entry tests a number of certificate attributes in such a way that the check will only pass if all of the configured attribute values match those in the client certificate. So, in effect, the attributes listed within a single attribute check are AND-ed together.

For example, imagine the following is configured as an entry in the **X.509 Attributes** table:

```
OU=Eng, O=Company Ltd
```

If the SOA Security Gateway receives a certificate with the following DName, this attribute check will pass because *all* the configured attributes match those in the certificate DName:

```
CN=User1, OU=Eng, O=Company Ltd, L=D4, S=Dublin, C=IE
CN=User2, OU=Eng, O=Company Ltd, L=D2, S=Dublin, C=IE
```

However, if the SOA Security Gateway receives a certificate with the following DName, the attribute check will fail because the attributes in the DName do not match *all* the configured ones (i.e. the "OU" attribute has the wrong value):

```
CN=User1, OU=qa, O=Company Ltd, L=D4, S=Dublin, C=IE
```

The **X.509 Attributes** table can contain several attribute check entries. In such cases, the attribute checks (i.e. the entries in the table) are OR-ed together, so that if any of the checks succeed, the overall **Certificate Attributes** filter succeeds.

So to summarize:

- Attribute values within an attribute check will only succeed if *all* the configured at-

tribute values match those in the DName of the client certificate.

- The filter will succeed if *any* of the attribute checks listed in the **X.509 Attributes** table succeed.

To configure a **Certificate Filter** complete the following fields:

Name:

Enter a name for the filter here.

X.509 Attributes:

To add a new X.509 attribute check, click the **Add button** button. In the **Add X.509 Attributes** dialog, enter a comma-seperated list of name-value pairs representing the X.509 attributes and their values, for example, "OU=dev,O=Company".

The new attribute check will appear in the **X.509 Attributes** table. Existing entries can be edited and deleted by clicking the **Edit** and **Remove** buttons respectively.

Retrieve Attribute from Database

Overview

The SOA Security Gateway can retrieve user attributes from a specified database. It can do this by running a SQL query on the specified database, or by invoking a stored procedure call.

Configuration

The following fields are available on the **Retrieve From Database** filter configuration screen.

Name:

Enter a name for this filter here.

User ID:

Select or enter the name of the message attribute that will contain the name of the user to look up in the database. For example, if the user name is stored under "admin" in the database, then the message attribute containing the value "admin" must be selected here. The SOA Security Gateway will then look up the database using this name.

Database Location:

The SOA Security Gateway will search the database selected here for the user's attributes. Connection details for the database can be configured by clicking on the **Add** button and completing the **Database Connection** dialog.

For more information on configuring the fields on this dialog, please refer to the Database Connection guide. Previously configured database connections can be edited and removed by selecting them in the dropdown and clicking the **Edit** and **Delete** buttons respectively.

Database Queries

The **Database Queries** table lists the currently configured SQL queries or stored procedure calls. These queries/calls retrieve certain user attributes from the database selected in the **Database Location** field above.

Existing queries can be edited and deleted by selecting them from the dropdown and clicking the **Edit** and **Delete** buttons respectively. For more information on how to con-

figure a **Database Query**, please refer to the Database Query Configuration guide.

Retrieve Attributes from Directory Server

Overview

The SOA Security Gateway can leverage an existing directory server by querying it for user profile data. The **Retrieve from Directory Server** filter can lookup a user,retrieve that user's attributes, and set them to the `attribute.lookup.list` message attribute.

General Configuration

The following fields are available on **Retrieve From Directory Server** filter configuration screen:

Name:

Enter a name for this filter here.

LDAP Directory:

The SOA Security Gateway will query the LDAP directory selected here for user attributes. To configure an LDAP directory, click the **Add/Edit** button. Take a look at the LDAP Configuration help page for more information on how to do this. An LDAP connection will be retrieved from a pool of connections at run-time.

Retrieve Unique User Identity

Use this section to select the user whose profile the SOA Security Gateway will look up in the directory server. The user ID can be taken from a message attribute or from an LDAP directory.

From Message Attribute:

Select this option if the user ID is stored in a message attribute. A user's credentials are stored in the `authentication.subject.id` message attribute after authenticating to the SOA Security Gateway and so this is the most likely attribute to enter in this field. Typically this will contain the Distinguished Name (DName) or username of the authenticated user. The name extracted from the selected message attribute will be used to query the directory server.

From LDAP Search:

Select this option to configure the SOA Security Gateway to retrieve the user's identity

from an LDAP search. Click the **Configure Directory Search** button to configure the search criteria to use to retrieve the user's identity. This option can be selected in cases where we the `authentication.subject.id` attribute has not been pre-populated by an authentication filter. In this case the user's unique DName must be retrieved from the LDAP repository.

Retrieve Attributes

This section instructs the SOA Security Gateway to search the LDAP tree according to certain conditions in order to locate a specific user profile. Once the appropriate profile has been retrieved, the SOA Security Gateway will extract the specified user attributes from it.

Base Criteria:

The value entered here tells the SOA Security Gateway where it should begin searching the LDAP directory. It is possible to enter a wildcard representing the value of a message attribute in this field. The two most likely message attributes to use here are the authenticated client's ID and DistinguishedName. The corresponding wildcard values are supplied by default:

- ${authentication.subject.id}
- ${authentication.subject.dname}

Search Filter:

This is the name given by the particular LDAP directory to the *User* class. This will depend on the type of LDAP directory that is configured. It is also possible to use wildcards here to represent the value of a message attribute. For example, the `user.role` attribute can be used to store the user class. The syntax for using the wildcard representing this attribute is as follows:

- (objectclass=${user.role})

Search Scope:

If the SOA Security Gateway retrieves a user profile node from the LDAP tree, the option selected here dictates the level that the SOA Security Gateway will search the node to. The options available are:

- Object level
- One level
- Sub-tree

Select the **Unique Result** checkbox to force the SOA Security Gateway to retrieve a unique user profile from the LDAP directory. This is useful in cases where the LDAP search has returned several profiles.

The **Attribute Name** table lists the attributes that the SOA Security Gateway will retrieve from the user profile. If no attributes are explicitly listed here, the SOA Security Gateway will extract all user attributes. In both cases, the retrieved attributes will be set to the `attribute.lookup.list` message attribute.

Click the **Add** button to add the name of an attribute to extract from the returned user profile. Simply enter the name of the attribute to extract from the profile in the **Attribute Name** field of the **Attribute Lookup** dialog.

Important Note:

It is important to note the following:

- If the search returns results for more that one user and the **Unique Result** option is enabled, an error will be generated. If this option is not enabled, all attributes will be merged.
- If an attribute is configured that does not exist in the repository, no error will be generated.
- If no attributes are configured, all attributes present for the user will be retrieved.

Retrieve Attribute from HTTP Header

Overview

The **Retrieve from HTTP Header** attribute retrieval filter can be used to retrieve the value of a HTTP header and set it to a message attribute. For example, it is possible to retrieve an X.509 certificate from a "USER_CERT" HTTP header and set it to the `authentication.cert` message attribute. This certificate can then be used by the filter's successors. The following HTTP request shows an example of such a header:

```
POST /services/getEmployee HTTP/1.1
Host: localhost:8095
Content-Length: 21
SOAPAction: HelloService
USER_CERT:  MIIEZDCCA0 ...9aKD1fEQgJ
```

It is also possible to retrieve a value from a named query string parameter and set this to the specified message attribute. The following example shows a request URL that contains a query string:

```
http://hostname.com/services/getEmployee?first=john&last=smith
```

In the above example the query string is "first=john&last=smith". As is clear from the example, query strings consist of attribute name-value pairs. Each name-value pair is separated by the '&' character.

Configuration

The following fields are available on the **Retrieve from HTTP Header** filter configuration screen:

Name:

Enter a name for this filter here.

HTTP Header Name:

Enter the name of the HTTP header that will contain the value that we want to set to the message attribute.

Base64 Decode:

Check this box if the extracted value should be base64 decoded before it is set to the message attribute.

Use Query String Parameters:

Check this box if the SOA Security Gateway should attempt to extract the **HTTP Header Name** from the query string parameters instead of from the HTTP headers.

Attribute ID:

Finally, select the attribute that will be used to store the value extracted from the request.

Retrieve Attribute from Message

Overview

The **Retrieve from Message** filter uses XPath expressions to extract the value of an XML element or attribute from the message and set it to an internal message attribute. It is also possible for the XPath expression to return a NodeList, and for the NodeList to be set to the specified message attribute.

Configuration

The following fields are available on the **Retrieve from Message** filter configuration screen:

Name:

Enter a name for this filter here.

Attribute Location:

Configure an XPath expression to retrieve the desired content.

Click the **Add** button to add an XPath expression. Existing expressions can be added and removed by clicking the **Edit** and **Remove** buttons respectively.

Extract the Content of the Node:

If this option is selected the content of the XML element or attribute in the message will be extracted and set to the selected message attribute.

Serialize All Nodes in the NodeList:

Use this option to select a NodeList to set to the message attribute. This option is useful for extracting `<Signature>`, `<Security>`, and `<UsernameToken>` blocks, as well as proprietary blocks of XML from messages.

Attribute ID:

The SOA Security Gateway will set the value of the message attribute selected here to the value extracted from the message. It is possible to enter a user-defined message attribute here as well.

Retrieve Attribute from User Store

Overview

The **User Store** stores a user's profile, including attributes relating to that user. After a user has successfully authenticated to the SOA Security Gateway, the **Retrieve From User Store** filter can retrieve attributes belonging to this user from the **User Store**. All attributes that are retrieved will be set to the `attribute.lookup.list`.

Configuration

The following fields are available on the **Retrieve From User Store** filter configuration screen:

Name:

Enter a name for this filter here.

User ID:

Select or enter the name of the message attribute that will contain the name of the user to look up in the **User Store**. For example, if the user name is stored as "admin", then the message attribute containing the value "admin" must be selected here. The SOA Security Gateway will then look up the **User Store** using this name.

Attributes:

Enter the list of attributes that the SOA Security Gateway should retrieve if it successfully looks up the user identified by the message attribute above. All attribute values will be stored in the `attribute.lookup.list` message attribute.

If no user attributes are specified here, the SOA Security Gateway will retrieve all the user's registered attributes and set them to the `attribute.lookup.list` attribute.

Attributes can be added by selecting the **Add** button. Similarly, existing attributes can be edited and removed by selecting the **Edit** and **Remove** buttons respectively.

Cache Attribute

Overview

The **Cache Attribute** filter allows you to configure what part of the message you want to cache. Typically, response messages are cached and so this filter is usually configured *after* the routing filters in a circuit. In this case the **content.body** attribute stores the response message body from the Web Service and so this message attribute should be selected in the **Attribute Name to Store** field.

For more information on how to configure this filter in the context of a "caching circuit", please refer to the Global Caches tutorial.

Configuration

Name:

Enter a name for this filter here.

Select Cache to Use:

The list of currently configured caches will be displayed in the tree. Select the name of the cache to store the attribute value in. Global caches (both local and distributed) can be added from the **Caches** top level node in the tree view of the SOA Security Gateway Management Console.

Attribute Key:

The value of the message attribute entered here acts as the key into the cache. In the context of a "caching circuit", it *must* be the same as the attribute specified in the **Attribute containing key** field on the **Is Cached?** filter.

Attribute Name to Store:

The value of the CA message attribute entered here will be cached in the cache specified in the **Cache to use** field above.

Create Key

Overview

The **Create Key** filter is used to identify the part of the message that is used to determine whether or not a message is unique. For example, it is possible to use the request message body to determine uniqueness so that if 2 successive identical message bodies are received by the SOA Security Gateway the response for the 2nd request will be taken from the cache. Other parts of the request can also be used to determine uniqueness, including HTTP headers, client IP address, client SSL certificate, and so on.

For more information on how to configure this filter in the context of a "caching circuit", please refer to the Global Caches tutorial.

Configuration

Name:

Enter a name for this filter here.

Attribute Name:

Enter the name of the CA message attribute to use to determine whether an incoming request is unique or not. So, for example, if **http.request.clientcert** (i.e. the client SSL certificate) is selected, the SOA Security Gateway will take a cached response for successive requests in which the client SSL certificate is the same.

Is Cached?

Overview

The **Is Cached?** filter looks up a named cache to see if a specified message attribute has already been cached. A message attribute - usually **message.key** - is used as the key to search for in the cache. If the lookup succeeds, the retrieved value overrides a specified message attribute, which is usually the **content.body** attribute.

For example, if a response message for a particular request has already been cached, the response message overrides the request message body so that it can be simply returned to the client using the **Reflect** filter.

For more information on how to configure this filter in the context of a "caching circuit", please refer to the Global Caches tutorial.

Configuration

Name:

Enter a name for this filter here.

Cache to Use:

The list of currently configured caches will be displayed in the tree. Select the name of the cache to lookup to find the attribute specified in the **Attribute containing key** field below. Global caches (both local and distributed) can be added from the **Caches** top level node in the tree view of the SOA Security Gateway Management Console.

Attribute containing key:

The message attribute entered here will be used as the key to lookup in the cache. In the context of a "caching circuit", the attribute entered here *must* be the same as the attribute specified in the **Attribute key** field on the **Cache Attribute** filter.

Overwrite Attribute Name if Found:

Usually the **content.body** is selected here so that value retrieved from the cache (which is usually a response message) overrides the request **content.body** with the cached response, which can then be returned to the client using the **Reflect** filter.

CRL File Filter

Overview

A CA may wish to publish a CRL (Certificate Revocation List) to a file. In such cases, the SOA Security Gateway can load the revoked certificates from the file-based CRL and validate user certificates against it.

The certificate of the CA that issued the CRL ***must*** be imported into the **Certificate Store** before this filter can work correctly.

The **CRL File Filter** requires the `certificates` message attribute to be set by a predecessor.

Configuration

Enter a name for the filter in the **Name** field of the **CRL File Filter** configuration screen. Click the **Load CRL** button to browse to the location of the CRL file. Read-only information regarding revocation dates will be displayed in the table on the **CRL File Filter** screen.

CRL LDAP Validation

Overview

A CRL (Certificate Revocation List) is a signed list indicating a set of certificates that are no longer considered valid (i.e. revoked certificates) by the certificate issuer. The SOA Security Gateway can query a CRL to find out if a given certificate has been revoked - if the certificate is present in the CRL, it should not be trusted.

In order to validate a certificate using a CRL lookup, the certificate's issuing CA's certificate should be trusted by the SOA Security Gateway. This is because for a CRL lookup, the CA's public key is needed to verify the signature on the CRL. The issuing CA's public key is not always included in the certificates that it issues, so it is necessary to retrieve it from the SOA Security Gateway's certificate store instead.

Configuration

The **Name** and **URL** of all currently configured LDAP directories are displayed in the table on the **CRL Certificate Validation** screen. The SOA Security Gateway will check the CRL of all LDAP directories with a checkbox beside them in order to validate the client certificate. The filter will fail as soon as the SOA Security Gateway determines that one of the CRLs has revoked the certificate.

To configure LDAP connection information, complete the following fields:

Name:

Enter a name for the filter here.

LDAP Configuration:

To add an LDAP directory to the table, click the **Add** button.

To configure the SOA Security Gateway to check the CRL of a configured LDAP directory, check the box next to the directory entry in the table on the main **Certificate Validation - CRL** screen.

Existing LDAP directory configurations can be edited and removed by clicking the **Edit** and **Remove** buttons respectively.

References

LDAP [http://www.cse.ohio-state.edu/cgi-bin/rfc/rfc1777.html]

Certificate Validity

Overview

The validity period of an X.509 certificate is encoded within the certificate. The **Certificate Validity** performs a simple check on a certificate to make sure that it has not expired.

By default, the **Certificate Validity** filter will search for the X.509 certificate in the `certificate` message attribute, which must be set by a predecessor filter in the circuit, e.g. by a **SSL Authentication** filter.

Configuration

Enter a name for the filter in the **Name** field of the **Certificate Validity** configuration screen. Enter or select the name of the message attribute that you expect to hold the certificate. The filter will check the validity of the certificate contained in this attribute. If no certificate is found, the filter will return an error.

Certificate Chain Check

Overview

It is a trivial task for a user to generate a structurally sound X.509 certificate and then use it to negotiate mutually authenticated connections to publicly available services. Clearly, this scenario is a security nightmare for IT administrators - we can't just allow any user to generate their own certificate and use it on the Internet.

For this reason, the SOA Security Gateway can establish the authenticity of the client certificate by ensuring that the certificate originated from a trusted source. To do this a server can perform a *certificate chain check* on the client certificate.

The main purpose of certificate chain validation is to ensure that a certificate has been issued by a trusted source. Typically, in a Public-Key Infrastructure (PKI), a Certificate Authority (CA) is responsible for issuing and distributing certificates. The whole infrastructure is based on the premise of *transitive trust* - if everybody trusts the CA, then everybody transitively trusts the certificates issued by that CA. If entities only trust certificates that have been issued by the CA, they can then reject certificates which have been self-generated by clients.

When a CA issues a certificate, it digitally signs the certificate and inserts a copy of its own certificate into it. This is called a certificate chain. Whenever an application (such as the SOA Security Gateway) receives a client certificate it can extract the issuing CA's certificate from it, and run a certificate chain check to determine whether or not it should trust the CA. If it trusts the CA, it will also trust the client certificate.

The SOA Security Gateway maintains a repository of both trusted CA certificates, known as the **Certificate Store**. In order to *trust* a certain CA, that CA's certificate must be imported into the **Certificate Store**.

Configuration

The table on the **Certificate Chain Check** screen lists the **DNames** of the certificates currently in the **Certificate Store**. Simply check the checkbox beside a CA in order for the SOA Security Gateway to consider it *trusted* for this certificate chain check. Multiple CAs can be selected.

References

Introduction to Public-Key Cryptography [http://docs.sun.com/source/816-6154-10/]

Find Certificate

Overview

The purpose of the **Find Certificate** filter is to locate a certificate and set it in the message for use by other certificate-based filters. Certificates can be extracted from the **User Store**, message attributes, HTTP headers, or attachments.

Configuration

By default, the SOA Security Gateway will store the extracted certificate in the `certificate` message attribute. However, it is possible to store the certificate in any message attribute, including any arbitrary attribute specified by the user, for example, a `user_certificate` attribute. The certificate can then be extracted from this attribute by a successor filter in the policy.

Name:

Enter a name for the filter in the **Name** field.

Attribute Name:

Enter or select the name of the message attribute to store the extracted certificate in.

Once the target message attribute has been selected, the next step is to specify the location of the certificate from one of the following options:

CA User:

Select a **User** whose certificate will be extracted from the **Certificate Store** and set to the message.

Certificate Store:

Click the **Select** button and select a certificate from the Trusted Certificate Store.

CA User or Wildcard:

This field represents an alternative way to specify what user's certificate is used. Either an explicitly named **User's** certificate will be used, or a wildcard can be specified to locate a **User** name or DName that can then be used to locate the certificate.

Wildcards are expressed by enclosing the message attribute that contains the user

name or DName in curly brackets, and prefixing this with the '$' sign. For example:

```
${authentication.subject.id}
```

This wildcard means that the SOA Security Gateway will use the certificate belonging to the subject of the authentication event in subsequent certificate-related filters. The certificate will be set to the `certificate` message attribute.

Using wildcards is a more generic way of locating certificates than specifying the **User** directly.

Message Attribute Name:

Enter the name of the message attribute that will contain the certificate.

HTTP Header Name:

Enter the name of the HTTP header that will contain the certificate.

Attachment Name:

Specify the name of the attachment (i.e. Content-Id) that will contain the certificate. It is possible to enter a wildcard in this field to represent the value of a message attribute.

OCSP Certificate Validation

Overview

OCSP (Online Certificate Status Protocol) is an automated certificate checking network protocol. The SOA Security Gateway can query an OCSP responder for the status of a certificate. The responder returns whether the certificate is still trusted by the CA that issued it.

In order to validate a certificate using an OCSP lookup, the issuing CA's certificate should be trusted by the SOA Security Gateway. This is because for an OCSP request, the protocol stipulates that the CA's public key must be submitted as part of the request. The issuing CA's public key is not always included in the certificates that it issues, so it is necessary to retrieve it from the SOA Security Gateway's certificate store instead. Take a look at the Certificate help page for more information on how to trust CA certificates.

Configuration

The table on the **Certificate Validation - OCSP** screen lists the currently available global OCSP Connections. OCSP Connections can be added globally from the tree view of the SOA Security Gateway Management Console, or by clicking the **Add** button on the **OCSP - Certificate Validation** filter screen.

The following fields can be configured on the **Certificate Validation - OCSP** dialog:

Name:

Enter a name for this OCSP configuration in this field.

URL Group:

Select a group of OCSP responders from the **URL Group** dropdown.

The SOA Security Gateway will attempt to connect to the OCSP responders in the selected group in a round-robin fashion. It will attempt to connect to the responders with the highest priority first, before connecting to responders with a lower priority. URL Groups can be added, edited, and removed by selecting the **Add**, **Edit**, and **Remove** buttons respectively. Take a look at the Configuring URL Groups help page for more information on adding and editing URL groups.

User Name:

Requests to OCSP responders can be signed by a **User** to whom the **Sign OCSP or**

XKMS Requests privilege has been assigned. Only those users who have been assigned this privilege will appear in the dropdown. For more information on assigning privileges to users, take a look at the Users guide.

Validate Response:

If the OCSP responders sign responses, check this checkbox to force the SOA Security Gateway to validate the signature on the response from the OCSP responder.

XKMS Certificate Validation

Overview

XKMS is an XML-based protocol for (amongst other things) establishing the trustworthiness of a certificate over the Internet. The SOA Security Gateway can query an XKMS responder to determine whether or not a given certificate can be trusted or not.

Configuration

The table on the **Certificate Validation - XKMS** screen lists the currently available global XKMS Connections. XKMS Connections can either be added from the "XKMS Connections" node beneath the "External Connections" tree item in the tree view of the SOA Security Gateway Management Console, or by clicking the **Add** button on this dialog.

The following fields can be configured on the **Certificate Validation - XKMS** screen.

Name:

Enter a name for this XKMS configuration in this field.

URL Group:

Select a group of XKMS responders from the **URL Group** dropdown.

The SOA Security Gateway will attempt to connect to the XKMS responders in the selected group in a round-robin fashion. It will attempt to connect to the responders with the highest priority first, before connecting to responders with a lower priority. URL Groups can be added, edited, and removed by selecting the **Add**, **Edit**, and **Remove** buttons respectively. Take a look at the Configuring URL Groups help page for more information on adding and editing URL groups.

User Name:

Requests to XKMS responders can be signed by a **User** to whom the **Sign OCSP or XKMS Requests** privilege has been assigned. Only those users who have been assigned this privilege will appear in the dropdown.

Content Type Filtering

Overview

The *SOAP Messages with Attachments* specification introduced a standard for transmitting arbitrary files along with SOAP messages as part of a multipart MIME message. In this way, both XML and non-XML data, including binary data, can be encapsulated in a SOAP message. The more recent DIME (Direct Internet Message Encapsulation) specification describes another way of packaging attachments with SOAP messages.

The SOA Security Gateway can accept or block multipart messages with certain MIME or DIME content types. For example, it is possible to configure a filter that blocks multipart messages that contain parts that are of type "image/jpeg".

Allow or Deny Types

The **Content Type Filtering** screen lists the content types that are allowed or denied by this filter.

Allow Content Types:

This option can be used if you wish to *accept* most content types, but only want to reject a few specific types. To allow or deny incoming messages based on their content types, complete the following simple steps:

1. Select the **Allow content types** radio button to allow multipart messages to be routed onwards. If you wish to simply allow all content types, there is no need to check any of the MIME types in the list.
2. To deny multipart messages with certain MIME or DIME types as parts, check the checkbox next to those types. Multipart messages containing parts of the MIME or DIME types selected here will be rejected.

Deny Content Types:

If you wish to *block* multipart messages containing most content types, but want to allow a small number of content types, then you should select this option. To reject multipart messages based on the content types of their parts:

1. Select the **Deny content types** radio button to reject multipart messages. If you wish to block all multipart messages, you do not need to check any of the MIME or DIME types in the list.

2. To allow messages with parts of a certain MIME or DIME type, check the checkbox next to those types. Multipart messages with parts of the MIME or DIME types selected here will be allowed. All other MIME or DIME types will be denied.

MIME and DIME types can be added by clicking the **MIME/DIME Registered Types** button. The next section describes how to add, edit, and remove MIME/DIME types.

Configuring MIME/DIME Types

The **MIME/DIME Settings** dialog allows you to configure new and existing MIME types. Once a type has been added, it is possible to configure the SOA Security Gateway to accept or block multipart messages with parts of this type.

Simply click the **Add** button to add a new MIME/DIME type or highlight a type in the table and select the **Edit** button to edit an existing type. To delete an existing type, select that type in the list and click the **Remove** button. Types can be added/edited using the **Configure MIME/DIME Type** dialog.

Enter a name for the new type in the **MIME or DIME Type** field, and the corresponding file extension in the **Extension** field.

References

SOAP Messages with Attachments [http://www.w3.org/TR/2000/NOTE-SOAP-attachments-20001211]

DIME [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/dimeindex.asp]

Validate Message Attributes

Overview

Filters that have been configured as predecessors of the **Validate Message Attributes** filter generate attributes and store them in the message. The **Validate Message Attributes** filter can the check the values of these message attributes. Regular expressions are used to check the values.

There are 2 sections that can be configured on this screen:

- A list of message attributes are configured so that are each checked against either a regular expression from the global **Whitelist** library or against a one-time expression. The regular expression check ensures that the value of the message attribute is acceptable.
- The second part of this filter runs each message attribute against the threatening content regular expressions contained within the global **Blacklist** library. These expressions scan for SQL injection attacks, ASCII control characters, XML entity expansion attacks, and many other common attack signatures.

The global **Whitelist** and **Blacklist** are available as a top-level item in the tree view of the SOA Security Gateway Management Console.

Message Attribute Regular Expressions

The table displays the ***whitelist*** of configured message attribute names together with the regular expressions that restrict their values. For this filter to run successfully, **all** of the configured attribute checks must have values matching the configured regular expressions.

The **Name** column gives the name of the attribute. The **Regular Expression** column gives the name of the regular expression that the SOA Security Gateway will use to restrict the value of the named attribute. As we will see a little later, a number of common regular expression are available for selection.

New regular expressions can be added, edited, and removed by selecting the **Add**, **Edit**, and **Delete** buttons respectively. Regular expressions are added/edited through the **Configure Regular Expression** interface.

This dialog allows you to configure regular expressions to restrict the values of message attributes. To configure such a regular expression, complete the following fields:

1. Enter the name of the message attribute in the **Name** field.

2. Specify whether this attribute is **Optional** or **Required** using the appropriate radio button. If this field is marked as "Required", the corresponding attribute **must** be present in the request. If it is not present, the filter will fail. If it is marked as "Optional", the attribute need not be present in order for the filter to pass.

3. The regular expression which is to restrict the value of the HTTP attribute can be entered either manually, or selected from the regular expression library of previously stored expressions. A number of common regular expressions - including alphanumeric values, dates, and email addresses - are provided for convenience. Enter or select an appropriate regular expression which will restrict the value of the specified attribute.

   Wildcards representing the values of message attributes can be used here to compare the value of a message attribute against another attribute. To do this, simply type the "$" character into the **Regular Expression** field to view a list of all available attributes. At run-time the wildcard will be replaced by the corresponding attribute value and compared to the message attribute whose value we are checking.

A regular expression can be added to the library by selecting the **Add/Edit** button. Simply enter a **Name** for the expression followed by the **Regular Expression** itself.

The **Advanced** section allows us to extract a portion of the header value which will then be run against the regular expression. The extracted substring can also be base64 decoded if necessary.

Threatening Content Regular Expressions

The purpose of the regular expressions entered in this section is to guard against the possibility of a message attribute containing malicious content. The table lists the ***blacklist*** regular expressions that will be run against all message attributes.

For example, to guard against a SQL DELETE attack, a regular expression could be written to identify SQL syntax and then added to this list. The **Threatening Content Regular Expressions** are listed in a table.

All of these expressions will be run against ***all*** the message attributes configured in the **Regular Expression** table above. If the expression matches ***any*** of the attribute values, the filter will fail.

Note:- It is important to note that if any regular expressions have been configured in the Message Attribute Regular Expressions section above, these expressions will be run ***before*** the Threatening Content Regular Expressions (TCRE) are run. So, for example, if we have already configured a regular expression to extract the base64 decoded value of an attribute value, then the TCRE will be run against this value as opposed to the at-

tribute value as it is stored in the message.

Threatening content regular expressions can be added using the **Add** button. Existing expressions can be edited or removed by highlighting them in the listbox and clicking the **Edit** and **Delete** buttons respectively.

The regular expressions themselves can either be entered manually or selected from the global **Blacklist** library of regular expressions. The library is pre-populated with a number of useful regular expressions that guard against common attack signatures. These include a number of expressions to guard against common SQL injection style attacks (e.g. "SQL Insert", "SQL Delete", etc), buffer overflow attacks (i.e. "Content longer than 1024 characters"), and the presence of control characters in attribute values (i.e. "ASCII Control Character").

Enter or select an appropriate regular expression that will scan all message attributes for threatening content. A regular expression can be added to the library by selecting the **Add/Edit** button. Simply enter a **Name** for the expression followed by the **Regular Expression** itself in the **Regular Expression** dialog.

Content Validation

Overview

This tutorial describes how the SOA Security Gateway can examine the contents of an XML message to ensure that it meets certain criteria. It uses boolean XPath expressions to evaluate whether or not a specific element or attribute contains has a certain value.

For example, it is possible to configure XPath expressions to make sure the value of an element matches a certain string, to check the value of an attribute is greater (or less) than a specific number, or that an element occurs a fixed amount of times within an XML body.

There are two ways to configure XPath expressions on this screen. Please click on the appropriate link below:

1. Manual XPath Configuration
2. XPath Wizard

1. Manual XPath Configuration

To manually configure a **Content Validation** rule using XPath:

1. Enter or select a name for the XPath filter rule in the **Name** dropdown.
2. Enter the XPath expression to use in the **XPath Expression** field.
3. In order to resolve any prefixes within the XPath expression, the namespace mappings (i.e. **Prefix**, **URI**) should be entered in the table.

As an example of how this screen should be configured, consider the following SOAP message:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="sample">
         ..............
         ..............
         ..............
         ..............
```

```
    </dsig:Signature>
  </soap:Header>
  <soap:Body>
    <prod:product xmlns:prod="http://www.company.com">
      <prod:name>SOA Product</prod:name>
      <prod:company>Company</prod:company>
      <prod:description>WebServices Security</prod:description>
    </prod:product>
  </soap:Body>
</soap:Envelope>
```

The following XPath expression evaluates to true if the <*company*> element contains the value "Company":

XPath Expression: //prod:company[text()='Company']

In this case it is necessary to define a mapping for the *prod* namespace as follows:

Prefix	***URI***
prod	http://www.company.com

Let's look at another example. This time the element that is to be examined by the XPath expression belongs to a default namespace. Consider the following SOAP message:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
  <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="sample">
      ..............
      ..............
      ..............
      ..............
  </dsig:Signature>
</soap:Header>
  <soap:Body>
    <product xmlns="http://www.company.com">
      <name>SOA Product</name>
      <company>Company</company>
      <description>Web Services Security</description>
    </product>
  </soap:Body>
</soap:Envelope>
```

The following XPath expression evaluates to true if the <*company*> element contains the value *Company*:

XPath Expression: //ns:company[text()='Company']

Since the <*company*> element actually belongs to the default (xmlns) namespace, i.e. *http://www.company.com*, it is necessary to make up an arbitrary prefix, *ns*, for use in the XPath expression and assign it to *http://www.company.com*. This is necessary to distinguish between potentially several default namespaces which may exist throughout the XML message. The following mapping illustrates this:

Prefix	***URI***
ns	http://www.company.com

2. XPath Wizard

The **XPath Wizard** assists administrators in creating correct and accurate XPath expressions. The wizard allows administrators to load an XML message and then run an XPath expression on it to determine what nodes are returned. To launch the **XPath Wizard**, click on the **XPath Wizard Button** on the **XPath Expression** dialog.

To use the XPath Wizard, simply enter (or browse to) the location of an XML file in the **File** field. The contents of the XML file will appear in the main window of the wizard. Enter an XPath expression in the **XPath** field and click the **Evaluate** button to run the XPath against the contents of the file. If the XPath expression returns any elements (or returns true), those elements will be highlighted in the main window.

If you are not sure how to write the XPath expression yourself, you can simply select an element in the main window. An XPath expression to isolate this element is automatically generated and displayed in the **Selected** field. If you wish to use this expression, select the **Use this path** button, and click **OK**.

References

XPath W3C Recommendation [http://www.w3.org/TR/xpath]

Namespaces in XML [http://www.w3.org/TR/REC-xml-names/]

XML Information Set [http://www.w3.org/TR/xml-infoset/]

HTTP Header Validation

Overview

The SOA Security Gateway can check HTTP header values for threatening content. This ensures that only properly configured name-value pairs appear in the HTTP request headers. *Regular expressions* are used to test the header values.

There are 2 sections that can be configured on this screen:

- A specified list of HTTP headers can be checked against a selected list of regular expressions from the global **Whitelist** library. These regular expressions ensure that the value of the HTTP header is acceptable.
- A list of regular expressions from the global **Blacklist** library of expressions can be run against all HTTP headers in the message. These regular expressions identify common attack signatures, such as SQL injection attacks, for example.

The global **Whitelist** and **Blacklist** are available as a top-level item in the tree view of the SOA Security Gateway Management Console.

HTTP Header Regular Expressions

The table displays the list of configured HTTP header names together with the ***whitelist*** of regular expressions that restrict their values. For this filter to run successfully, **all** of the required headers must be present in the request, and **all** must have values matching the configured regular expressions.

The **Name** column gives the name of the HTTP header. The **Regular Expression** column gives the name of the regular expression that the SOA Security Gateway will use to restrict the value of the named HTTP header. As we will see a little later, a number of common regular expression are available for selection.

New regular expressions can be added, edited, and removed by selecting the **Add**, **Edit**, and **Delete** buttons respectively. Regular expressions are added/edited through the **Configure Regular Expression** interface.

This dialog allows you to configure regular expressions to restrict the values of HTTP headers. To configure such a regular expression, complete the following fields:

1. Enter the name of the HTTP header in the **Name** field.
2. Specify whether this header is **Optional** or **Required** using the appropriate radio

button. If this field is marked as "Required", the corresponding header **must** be present in the request. If it is not present, the filter will fail. If it is marked as "Optional", the header need not be present in order for the filter to pass.

3. The regular expression that is to restrict the value of the HTTP header can be entered either manually, or selected from the global **Whitelist** library of regular expressions. A number of common regular expressions - including alphanumeric values, dates, and email addresses - are provided for convenience. Enter or select an appropriate regular expression which will restrict the value of the specified header.

   Wildcards representing the values of message attributes can be used here to compare the value of a HTTP header against the value contained in a message attribute. To do this, simply type the "$" character into the **Regular Expression** field to view a list of all available attributes. At run-time the wildcard will be replaced by the corresponding attribute value and then compared to the HTTP header that we want to check the value of.

A regular expression can be added to the library by selecting the **Add/Edit** button. Simply enter a **Name** for the expression followed by the **Regular Expression** itself.

The **Advanced** section allows us to extract a portion of the header value which will then be run against the regular expression. The extracted substring can also be base64 decoded if necessary.

This section is specifically aimed towards HTTP Basic authentication headers, which consist of the "Basic " prefix followed by the base64 encoded username and password. The following is an example of the HTTP Basic authentication header:

```
Authorization: Basic dXNlcjp1c2Vy
```

Clearly it is the base64 encoded portion of the header value that we are interested in running the regular expression against. We can extract this by specifying the string that occurs directly before the substring we want to extract, together with the string that occurs directly after the substring.

To extract the base64 encoded section of the *Authorization* header above, simply enter "Basic" in the **Start substring** field and leave the **End substring** field blank to signify that we wish to extract the entire remainder of the header value.

Note:- It is important to note that the start and end substrings must be selected to ensure that the exact substring is extracted. For example, in the HTTP Basic example

above, "Basic " (including trailing space) should be entered in the **Start substring** field, and **not** "Basic" (with no trailing space).

By specifying the correct substrings, we are left with the base 64 encoded header value, i.e. "dXNlcjp1c2Vy". However, we still need to base64 *decode* it before we can run a regular expression on it. To do this, make sure to check the **Base64 decode** checkbox. The base64 decoded header value is "user:user", which conforms to the standard format of the *Authorization* HTTP Header. It is this value that we want to run the regular expression against.

The next example shows an example of a HTTP Digest authentication header:

```
Authorization: Digest username="user", realm="ca.com", qop="auth",
algorithm="MD5", uri="/editor", nonce="Id-00000109924ff10b-0000000000000091",
nc="1", cnonce="ae122a8b549af2f0915de868abff55bacd7757ca",
response="29224d8f870a62ce4acc48033c9f6863"
```

It is possible to extract single values from the header value. For example, to extract the "realm" field, simply enter, realm=" (including the " character), in the **Start substring** field and, ", in the **End substring** field. This leaves us with "ca.com" to run the regular expression against. In this case, there is no need to base64 decode the extracted sub-string.

Note that if both **Start substring** and **End substring** fields are left blank, the regular expression will be run against the entire header value. Furthermore, if both fields are left blank and the **Base64 decode** box is checked, the entire header value will be base64 encoded before the regular expression is run against it.

While the above example deals specifically with the HTTP authentication headers, the interface is generic enough to allow you to extract a substring from other header values as well.

Threatening Content Regular Expressions

The purpose of the regular expressions entered in this section is to guard against the possibility of a HTTP header containing malicious content. The table lists the ***blacklist*** of regular expressions to run to ensure that the header values do not contain threatening content.

For example, to guard against a SQL DELETE attack, a regular expression could be written to identify SQL syntax and then added to this list. The **Threatening Content Regular Expressions** are listed in a table.

All of these expressions will be run against ***all*** HTTP header values in an incoming re-

quest. If the expression matches ***any*** of the values, the filter will fail.

Note:- It is important to note that if any regular expressions have been configured in the HTTP Header Regular Expressions section above, these expressions will be run ***before*** the Threatening Content Regular Expressions (TCRE) are run. So, for example, if we have already configured a regular expression to extract the base64 decoded value of the *Authentication* header value as in the example above, then the TCRE will be run against this value as opposed to the attribute value that appears in the query string.

Threatening content regular expressions can be added using the **Add** button. Existing expressions can be edited or removed by highlighting them in the listbox and clicking the **Edit** or **Delete** buttons respectively.

The regular expressions themselves can either be entered manually or selected from the global **Blacklist** library of threatening content regular expressions. This library is pre-populated with a number of regular expressions that scan for common attack signatures. These include a number of expressions to guard against common SQL injection style attacks (e.g. "SQL Insert", "SQL Delete", etc), buffer overflow attacks (i.e. "Content longer than 1024 characters"), and the presence of control characters in attribute values (i.e. "ASCII Control Character").

Enter or select an appropriate regular expression that will restrict the value of the specified HTTP header. A regular expression can be added to the library by selecting the **Add/Edit** button. Simply enter a **Name** for the expression followed by the **Regular Expression** itself.

References

Regular Expressions [http://www.opengroup.org/onlinepubs/007908799/xbd/re.html]

Maximum Messages

Overview

The **Maximum Messages** filter can protect a Web Service or SOA (Service Oriented Architecture) from message flooding. The filter can be configured to only allow a certain number of messages from a specified client in a given timeframe through to the protected system.

If the number of messages exceeds the specified limit, the filter will fail for the excess messages. It is important to note that the filter will still succeed for incoming messages that meet the specified constraints. So, for example, if the filter is configured to allow 20 messages through per second, it will fail for the 21st message, but pass for the first 20 incoming messages.

The SOA Security Gateway's behavior in the case of a breach in the configured constraints is determined by the filter that is next in the failure path for the **Maximum Messages** filter in the policy. Typically, an **Alert**, **Trace**, or **Log** filter will be configured as the successor filter in the failure path in the policy.

An example use-case of this filter would be to protect a Web Service that can only handle a maximum of 20 messages per client per second. If the filter detects a higher number of incoming requests, it will block the messages.

Configuration

Complete the following fields to configure the SOA Security Gateway to only allow a certain number of messages in a given time interval:

Name:

Enter a name for the filter here.

Number of Messages:

If the SOA Security Gateway receives more than this number of messages during the time interval specified in the field below, the filter will fail. Otherwise the filter will pass.

Time Period:

If the SOA Security Gateway receives more than the number of messages specified in the above field in the time interval specified here, the filter will fail. Otherwise the filter will pass. The actual time period will depend on the value selected below, i.e. seconds, minutes, hours, days, or weeks. So, for example, if "10" is entered as the **Time Period**

and "Minutes" is selected from the **Time Period Unit** dropdown below, the time period will last 10 minutes.

Time Period Unit:

The unit the time period is measured in must be selected from the following options: "Second", "Minute", "Hour", "Day", or "Week". The value selected here together with that entered above will determine the actual time period.

Note that when one of "Second", "Minute", or "Hour" is selected here, the **Time Period Commences on Hour/Day** fields need not be configured. The time period will commence when a message is received and will last for the time period, e.g. 10 minutes. Once this time period is up the message count will be reset and the counter will start again when another message is received. See the sections below for how the time period is measured when the **Time Period Unit** is set to "Day" or "Week".

Time Period Commences on Hour:

This field must be configured if you select either "Day" or "Week" as the **Time Period Unit** above. For example, if you select "Day" above and enter "00:00" in this field, this means that only the specified number of messages can be received in a 1 day period starting from midnight tonight until midnight the next day.

Time Period Commences on Day:

This field must only be configured if you select "Week" as the **Time Period Unit** above. For example, if you select "Week" and "00:00" in the fields above and then enter "Sunday" here, this means that the time period will commence next Sunday at midnight and will last for 1 week exactly. The time period will be reset on midnight of the next Sunday.

Track per Key:

Check this box if you wish to configure the SOA Security Gateway to keep track of request messages based on a specific key value. In other words, if more messages that match this key are received than are allowed, the filter will fail.

Key Value:

The **Track per Key** toggle above can be used to perform message filtering based on a particular key. This key can be used to lookup entries in the cache selected below. The key entered here can be a combination of a fixed string value and/or a CA message attribute. For example, the following key could be used to keep track of the number of times a particular URI is requested:

```
MSG_COUNT-${http.request.uri}
```

Cache to Use:

In cases where multiple SOA Security Gateways are deployed for load balancing purposes and you want to maintain a single count of all messages processed by all the SOA Security Gateway instances, you can configure a distributed cache to cache request messages.

For example, let's assume the intention is to prevent a burst of more than 50 messages per second from reaching the back-end Web Service. Let's also assume that a load balancer has been deployed in front of 2 instances of the SOA Security Gateway and is round-robining requests between these 2 instances. By caching request messages in a global distributed cache, which is inherently replicated across all SOA Security Gateway instances, it is possible for the **Maximum Messages** filter to compute the total number of messages in the distributed cache and hence, the total number of messages processed by all SOA Security Gateway instances.

The **Maximum Messages** filter is configured to use the pre-configured **Local maximum messages** distributed cache by default. It is, of course, possible to configure more caches using the Global Cache interface.

Message Size

Overview

It is sometimes useful to filter incoming messages based, not only on the content of the message, but on external characteristics of the message. To this end, the SOA Security Gateway can be configured to reject messages that are greater or less than a specified size.

Configuration

To configure the SOA Security Gateway to block messages of a certain size, complete the following fields:

- Enter the size (in bytes) of the smallest message that should be processed in the **At least** field. Messages smaller than this size will be rejected.
- Enter the size (in bytes) of the largest message that should be processed in the **At most** field. Messages larger than the size entered here will be rejected.
- The **Use in Size Calculation** options are used to specify the portion of the message that is to be used when calculating the size of the message.
    - If the **Root body only** option is selected, the SOA Security Gateway will calculate the size of the message body excluding all other MIME parts, i.e. attachments.
    - If the **Attachments only** option is selected, the SOA Security Gateway will only calculate the size of all attachments to the message. It will exclude the size of the root body payload from its calculation.
    - Finally, if the **Root body and attachments** option is selected, the SOA Security Gateway will include the root body together with all other MIME parts when it calculates the size of the message.

It is important to note the following:

- The message size measured by the SOA Security Gateway does ***not*** include HTTP headers.

Query String Validation

Overview

The SOA Security Gateway can check the request *query string* to ensure that only properly configured name and value pairs appear. *Regular expressions* are used to test the attribute values.

There are 2 sections that can be configured on this screen:

- A specified list of query string names can be checked against a selected list of regular expressions from the global **Whitelist** library. These regular expressions ensure that the corresponding value is acceptable.
- A list of regular expressions from the global **Blacklist** library of expressions can be run against all query string names and values. These regular expressions identify common attack signatures, such as SQL injection attacks, for example.

Query String Overview

The request query string is that portion of the URL that comes after the '?' character, and contains the request parameters. It is typically used for HTTP GET requests in which form data is submitted as name-value pairs on the URL. This contrasts to the HTTP POST method where the data is submitted in the body of the request. The following example shows a request URL that contains a query string:

```
http://hostname.com/services/getEmployee?first=john&last=smith
```

In the above example the query string is "first=john&last=smith". As is clear from the example, query strings consist of attribute name-value pairs. Each name-value pair is separated by the '&' character.

The **Query String Validation** filter can also operate on the form parameters submitted in a HTTP Form POST. Instead of encoding the request parameters in the query string, the client uses the 'application/x-www-form-urlencoded' content-type, and submits the parameters in the HTTP POST body, for example:

```
POST /services/getEmployee HTTP/1.1
Host: localhost:8095
Content-Length: 21
SOAPAction: HelloService
Content-Type: application/x-www-form-urlencoded

first=john&last=smith
```

If the SOA Security Gateway receives a HTTP request body such as this, the **Query String Validation** filter can validate the form parameters.

Query String Attribute Regular Expressions

The table displays the list of configured query string names together with the ***whitelist*** of regular expressions that restrict their values. For this filter to run successfully, **all** of the required attributes must be present in the request, and all must have the correct value.

The **Name** column gives the name of the query string attribute. The **Regular Expression** column gives the name of the regular expression that the SOA Security Gateway will use to restrict the value of the named query string attribute. As we will see a little later, a number of common regular expression are available for selection from the global **Whitelist** library of regular expressions.

If the **Allow unspecified names** checkbox has been checked, additional un-named query string attributes will not be filtered by the SOA Security Gateway. This is useful in cases where we are interested in filtering the content of only a small number of query string attributes, for example, but the request may contain many attributes. In such cases, it is only necessary to filter those few attributes, and by checking this checkbox, the SOA Security Gateway will ignore all other query string attributes.

New regular expressions can be added, edited, and removed by selecting the **Add**, **Edit**, and **Delete** buttons respectively. Regular expressions are added/edited through the **Configure Regular Expression** interface.

This dialog allows you to configure regular expressions to restrict the values of request query string attributes. To configure such a regular expression, simply complete the following fields:

1. Enter the name of the query string attribute in the **Name** field.
2. Specify whether this request parameter is **Optional** or **Required** using the appropriate radio button. If this field is marked as "Required", the corresponding parameter name **must** be present in the request. If it is not present, the filter will fail. If

it is marked as "Optional", the attribute need not be present in order for the filter to pass.

3. The regular expression that is to restrict the value of the query string attribute can be entered either manually, or selected from the global **Whitelist** library of regular expressions. A number of common regular expressions are provided for convenience. For example, regular expressions are supplied for alphanumeric values, dates, and email addresses.

   Wildcards representing the values of message attributes can be used here to compare the value of the query string attribute against the value contained in a message attribute. To do this, simply type the "$" character into the **Regular Expression** field to view a list of all available attributes. At run-time the wildcard will be replaced by the corresponding attribute value and then compared to the query string attribute that we want to check the value of.

Enter or select an appropriate regular expression which will restrict the value of the named attribute. A regular expression can be added to the library by selecting the **Add/Edit** button. Simply enter a **Name** for the expression followed by the **Regular Expression** itself.

The **Advanced** section allows us to extract a portion of the query string attribute value which will then be run against the regular expression. The extracted substring can also be base64 decoded if necessary.

The following is an example of a URL containing a query string. The value of the "password" attribute is base64 encoded and must be extracted from the query string and then decoded before it can be run against the regular expression.

```
http://ca.com/services?username=user&password=dXNlcg0K&dept=eng
```

We can extract the encoded value of the "password=" attribute value by specifying the string that occurs directly before the substring we want to extract, together with the string that occurs directly after the substring. To do this, simply enter "password=" (without quotes) in the **Start substring** field and "&" (again without quotes) in the **End substring** field.

Note:- It is important to note that the start and end substrings must be selected to ensure that the exact substring is extracted. For example, in the example above, "password= " (including the equals sign) should be entered in the **Start substring** field, and **not** "password" (without the equals sign).

By specifying the correct substrings, we are left with the base 64 encoded attribute value, i.e. "dXNlcg0K". However, we still need to base64 *decode* it before we can run a regular expression on it. To do this, make sure to check the **Base64 decode** checkbox. The base64 decoded password value is simply "user". It is this value that we want to run the regular expression against.

Note that if both **Start substring** and **End substring** fields are left blank, the regular expression will be run against the entire attribute value. Furthermore, if both fields are left blank and the **Base64 decode** box is checked, the entire attribute value will be base64 encoded before the regular expression is run against it.

Threatening Content Regular Expressions

The purpose of the regular expressions entered in this section is to guard against the possibility of a query string attribute containing malicious content. The table lists the ***blacklist*** of regular expressions to run to ensure that the query string values do not contain threatening content.

For example, to guard against a SQL DELETE attack, a regular expression could be written to identify SQL syntax and then added to this list. The **Threatening Content Regular Expressions** are listed in a table.

All of these expressions will be run against ***all*** attribute values in the query string. If the expression matches ***any*** of the values, the filter will fail.

Note:- It is important to note that if any regular expressions have been configured in the section above, these expressions will be run ***before*** the Threatening Content Regular Expressions (TCRE) are run. So, for example, if we have already configured a regular expression to extract the base64 decoded value of the "password" query string attribute as in the example above, then the TCRE will be run against this value as opposed to the attribute value that appears in the query string.

Threatening content regular expressions can be added using the **Add** button. Existing expressions can be edited or removed by highlighting them in the listbox and clicking the **Edit** and **Delete** buttons respectively.

The regular expressions themselves can either be entered manually or selected from the global **Blacklist** library of threatening content regular expressions. This library is pre-populated with a number of regular expressions that guard against common attack signatures. These include a expressions to guard against common SQL injection style attacks (e.g. "SQL Insert", "SQL Delete", etc), buffer overflow attacks (i.e. "Content longer than 1024 characters"), and the presence of control characters in attribute values (i.e. "ASCII Control Character").

Enter or select an appropriate regular expression that will restrict the value of the specified HTTP header. A regular expression can be added to the library by selecting the **Add/Edit** button. Simply enter a **Name** for the expression followed by the **Regular**

Expression itself.

References

Regular Expressions [http://www.opengroup.org/onlinepubs/007908799/xbd/re.html]

Regular Expression Configuration

Overview

Regular expressions are strings (or patterns) that match a **set** of strings or patterns. For example, it is possible to write regular expressions that match all email addresses, alphabetic characters only, or even IP addresses.

The SOA Security Gateway can use such regular expressions to ensure that HTTP header values and/or query string attribute values conform to configured patterns. For example, it is possible to configure a regular expression to make sure that the value of a "User" HTTP header or a "User" query string attribute is a valid email address. To illustrate this, let's look at how the regular expression can be applied to both cases:

HTTP Header:

```
POST /services/getEmployee HTTP/1.1
Content-Type: text.html
User: user@ca.com
```

Query String:

```
http://hostname.com/services/getEmployee?User=user@ca.com
```

The following regular expression can be used to ensure that both the header and attribute values conform to the standard email format:

```
\b[a-zA-Z0-9._%-]+@[a-zA-Z0-9._%-]+\.[a-zA-Z]{2,4}\b
```

To learn how to configure regular expressions for the respective request parameters, click on the appropriate link below:

- Query String Validation
- HTTP Header Validation

References

Regular Expressions [http://www.opengroup.org/onlinepubs/007908799/xbd/re.html]

Content Filtering - Request Parameters

Overview

The SOA Security Gateway can check both the request *Query String* and HTTP header values for threatening content. This ensures that only properly configured names and values appear in the Query String and HTTP request headers. *Regular expressions* are used to test the attribute values.

Query String Overview

The request Query String is that portion of the URL that comes after the '?' character, and contains the request parameters. It is typically used for HTTP GET requests in which form data is submitted as name-value pairs on the URL. This contrasts to the HTTP POST method where the data is submitted in the body of the request. The following example shows a request URL that contains a Query String:

```
http://hostname.com/services/getEmployee?first=john&last=smith
```

In the above example the Query String is "first=john&last=smith". As is clear from the example, Query Strings consist of attribute name-value pairs. Each name-value pair is separated by the '&' character.

1. Query String/HTTP Header Attribute Regular Expressions

The table contains the list of attribute names, together with their legitimate values, that must be received either as part of the request Query String or as a HTTP request header. For this filter to run successfully, **all** of the required attributes must be present in the request, and must have the correct value.

The **Name** column gives the name of the HTTP header field or Query String attribute. In order for this filter to succeed, **all** named headers and Query String attributes **must** be present in the incoming request.

The **Type** column indicates whether the request parameter is a HTTP header or a Query String attribute.

The **Regular Expression** column gives the name of the regular expression that the SOA Security Gateway will use to restrict the value of the named request parameter.

The value of the named request parameter **must** comply with the constraints imposed by the regular expression. As we will see a little later, a number of common regular expression are available for selection.

The **Required** field indicates whether the attribute must be present or not. If this field is marked as "Yes", the corresponding parameter name **must** be present in the request. If it is not present, the filter will fail. If it is marked as "No", the attribute need not be present in order for the filter to pass.

If the **Allow unspecified attributes** checkbox has been checked, additional un-named header names and Query String attributes will not be filtered by the SOA Security Gateway. This is useful in cases where we are interested in filtering the content of only a small number of Query String attributes, for example, but the request may contain many attributes. In such cases, it is only necessary to filter those few attributes, and by checking this checkbox, the SOA Security Gateway will ignore all other Query String attributes.

New regular expressions can be added, edited, and removed by selecting the **Add**, **Edit**, and **Delete** buttons respectively. Regular expressions are added/edited through the **Configure Query String or HTTP Header** interface.

This dialog allows you to configure regular expressions to restrict the values of request Query Strings or HTTP headers. To configure such a regular expression, simply complete the following fields:

1. The regular expression can be applied to a Query String attribute or to the value of a named HTTP header. Select the appropriate type of request parameter using the **Type** field.
2. Enter the name of the Query String attribute or HTTP header in the **Name** field.
3. Specify whether this request parameter is **Optional** or **Required** using the appropriate radio button.
4. The regular expression which is to restrict the value of the request parameter can be entered either manually, or selected from the regular expression library of previously stored expressions. A number of common regular expressions are provided for convenience. For example, regular expressions are supplied for alphanumeric values, dates, and email addresses.

   Enter or select an appropriate regular expression which will restrict the value of the specified request parameter. A regular expression can be added to the library by selecting the **Add/Edit** button. Simply enter a **Name** for the expression followed by the **Regular Expression** itself.

2. Threatening Content Regular Expressions

The purpose of the regular expressions entered in this section is to guard against the possibility of a HTTP header or Query String attribute containing malicious content. For example, an expression could be written to identify SQL syntax, thus preventing SQL injection attacks.

All of these expressions will be run against ***all*** attribute values in the QueryString and/or HTTP headers. If the expression matches ***any*** of the values, the filter will fail.

Threatening content regular expressions can be added using the **Add** button. Existing expressions can be edited or removed by highlighting them in the listbox and clicking the **Edit** or **Delete** buttons respectively.

To configure a threatening content regular expression, complete the following fields.

- The specified regular expression can be applied to HTTP headers only, Query String attributes only, or both HTTP headers and Query String attributes. Select the appropriate option from the **Type** dropdown.
- The regular expressions themselves can either be entered manually or selected from a library of previously stored threatening content regular expressions. A number of useful regular expressions which guard against common attacks, including SQL injection and buffer overflow style attacks, are provided for convenience. For example, a number of expressions are supplied which guard against some common SQL injection style attacks (e.g. "SQL Insert", "SQL Delete", etc), buffer overflow attacks (i.e. "Content longer than 1024 characters"), and the presence of control characters in parameter values (i.e. "ASCII Control Character").

  Enter or select an appropriate regular expression which will restrict the value of the specified request parameter. A regular expression can be added to the library by selecting the **Add/Edit** button. Simply enter a **Name** for the expression followed by the **Regular Expression** itself.

References

Regular Expressions [http://www.opengroup.org/onlinepubs/007908799/xbd/re.html]

Schema Validation

Overview

The SOA Security Gateway can check that XML messages conform to the structure or format expected by the Web Service by validating those requests against XML Schemas. An XML Schema precisely defines the elements and attributes that constitute an instance XML document. It also specifies the data types of these elements to ensure that only appropriate data is allowed through to the Web Service.

For example, an XML Schema might stipulate that all requests to a particular Web Service must contain a <name> element, which contains at most a ten character string. If the SOA Security Gateway receives a message with an improperly formed <name> element, it will reject the message.

The **Schema Validation** filter can be found under the "Content Filtering" category of filters in the SOA Security Gateway Management Console. Drag and drop the filter onto the policy where you want to perform schema validation. The **Schema Validation** filter dialog has 3 tabs, each of which will be explained in the following sections.

1. Schema to Use

Schemas from either the global **Schema Cache** or the **Web Services Repository** can be selected in this filter in order to validate messages. Take a look at the following tutorials to find out how to import schemas into these global stores so that they can be selected in the **Schema Validation** tutorial:

- Global Schema Cache
- Web Service Repository

The tree hierarchy on the **Schema to Use** tab contains 2 top-level nodes: one for the **Schema Cache** and the other for the **Web Services Repository**. When these nodes are expanded the XML Schemas contained within them are displayed in a tree structure. Each schema has a checkbox beside it, which can be checked in order to select the schema to use to validate the incoming message.

It is worth noting at this point that if you have a WSDL file that contains an XML Schema, and you want to use this schema to validate the message, you can import the WSDL file into the **Web Services Repository**. The **WSDL Import Wizard** has an option to automatically create a **Schema Validation** filter and incorporate it into the auto-generated policy. In this case, the top-level schema in the WSDL, which will be imported into the **Web Services Repository**, will be selected by default in the filter. In

this way, if the schema imports other schemas, they will be available to the filter at run-time when validating the message. For more information on how to import WSDL files in this manner, please refer to the Web Services Repository help page.

2. Part of Message to Match

A portion of the XML message can be extracted using an XPath expression. The SOA Security Gateway can then validate this portion against the specified XML Schema. For example, administrators may only want to validate the SOAP Body part of a SOAP message. In this case, they should enter (or select) an XPath expression that identifies the SOAP Body of the message. This portion should then be validated against an XML Schema which defines the structure of the SOAP Body for that particular message.

Click on the **Add** or **Edit** buttons to add or edit an XPath expression using the **Enter XPath Expression** dialog. Expressions can be removed by selecting the expression in the **XPath Expression** dropdown and clicking the **Delete** button.

On the **Enter XPath Expression** dialog, there are two ways to configure XPath expressions. Please click on the appropriate link below for more information:

1. Manual Configuration
2. XPath Wizard

The **XPath Wizard** assists administrators in creating correct and accurate XPath expressions. The wizard allows administrators to load an XML message and then run an XPath expression on it to determine what nodes are returned.

3. Advanced

When the **Allow RPC Schema Validation** checkbox is checked, the filter will make a *best attempt* to validate an RPC encoded SOAP message. An RPC encoded message is defined in the WSDL as having an operation with the following characteristics:

- The **style** attribute of the <soap:operation> element is set to "document".
- The **use** attribute of the <soap:body> element is set to "rpc".

Take a look at Section 3.5 [http://www.w3.org/TR/wsdl#_soap:body] of the WSDL specification, which explains the possible values for these attributes.

The problem with RPC encoded SOAP messages in terms of schema validation is that the schema contained within the WSDL file does not necessarily fully define the format of the SOAP message, unlike with "document-literal" style messages, for example. With an RPC encoded operation, the format of the message can be defined by a combination

of the SOAP operation name, WSDL message parts, and schema-defined types. As a result, the Schema extracted from a WSDL file may not be able to validate a message.

Another problem with RPC encoded messages is that type information is included in each element that appears within the SOAP message. In order for such element definitions to be validated by a schema, the type declarations must be removed, which is precisely what the **Schema Validation** filter does if the checkbox is checked on this tab. It removes the type declarations and then makes a *best attempt* to validate the message.

However, as explained earlier, if some of the elements in the SOAP message are actually taken from the WSDL file as opposed to the Schema (for example, when the SOAP operation name in the WSDL file is used as the wrapper element beneath the SOAP Body as opposed to a schema-defined type), the schema will **not** be able to validate the message.

References

XML Schema Part 0: Primer, W3C Recommendation [http://www.w3.org/TR/xmlschema-0/]

XML Schema Part 1: Structures, W3C Recommendation [http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/]

XML Schema Part 2: Datatypes, W3C Recommendation [http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/]

Namespaces in XML [http://www.w3.org/TR/REC-xml-names/]

XPath W3C Recommendation [http://www.w3.org/TR/xpath]

WSDL 1.1 Specification [http://www.w3.org/TR/wsdl]

Validate Timestamp

Overview

The **Validate Timestamp** filter can be used to validate a timestamp that has been stored in a message attribute by a previous filter in a policy.

For example, it is possible to extract the value of a "wsu:Created" element from a WS-Security token and store it in a "created" attribute using the **Attributes - Retrieve from Message** filter. The **Validate Timestamp** filter can then be used to ensure that the "created" timestamp is not *after* the current time.

Similarly, the **Attributes - Retrieve from Message** filter can be used to extract the value of the "wsu:Expires" element and store it in a "timestamp" message attribute. The **Validate Timestamp** filter can be used to check that the timestamp is not *before* the current time.

So essentially what we are doing is to make sure that the current time is between the "Created" time and the "Expires" time. When taking into account the drift time (to resolve discrepancies between the clock times on the machine that generated the timestamp and the machine running the SOA Security Gateway), we are really ensuring that the current time is after the "Created" time minus the drift time, and before the "Expires" time plus the drift time, i.e. that the current time is within the timeframe:

```
Created time - drift --> Expires + drift
```

Configuration

Complete the following fields to configure the SOA Security Gateway to validate a timestamp that has been stored in a message attribute:

Name:

Enter a name for the filter here.

Attribute Containing Timestamp:

Enter the name of the message attribute that contains the value of the timestamp in this field. You will need to configure a predecessor of this filter to extract the timestamp

from the message and store it in the specified attribute, e.g. the **Attributes - Retrieve from Message** filter.

Format of Timestamp:

Enter the format of the timestamp that will be contained within the specified message attribute.

Drift (secs):

Specify the drift time to use when determining whether or not the current time falls within a certain time interval. The drift time can be used to account for differences in the clock times of the machine running the SOA Security Gateway and the machine on which the timestamp was generated.

Current Time Must be After the Timestamp:

The current time must be *after* the timestamp (plus or minus the drift time) in order for it to be considered valid.

Current Time Must be Before the Timestamp:

The current time must be *before* the timestamp (plus or minus the drift time) in order for it to be considered valid.

XML Complexity

Overview

Parsing XML documents is a notoriously processor-intensive activity. This fact can be exploited by hackers by sending large and complex XML messages to Web Services in a type of denial-of-service attack in the hope of overloading them. The **XML Complexity** filter can protect against such attacks by performing the following checks on an ingress XML message:

- Checking the total number of nodes contained within the XML message.
- Ensuring that the message does not contain deeply nested levels of XML nodes.
- Making sure that elements within the XML message can only contain a specified maximum number of child elements.
- Making sure that each element can have a maximum number of attributes.

By performing these checks, the SOA Security Gateway can protect back-end Web Services from having to process large and potentially complex XML messages.

Configuration

The **XML Complexity** filter should be configured as the 1st filter in the policy that processes the XML body. This gives the filter the opportunity to block any excessively large or complex XML message *before* any other filters attempt to process the XML.

Complete the following fields to configure the **XML Complexity** filter:

Name:

Enter a name for this filter.

Maximum Total Number of Nodes:

Specify the maximum number of nodes that you want to allow in an XML message. Note that this number does not include text nodes or comments.

Maximum Number of Levels of Descendant Nodes:

Enter the maximum number of descendant nodes that an element is allowed to have.

Again, this number does not include text nodes or comments.

Maximum Number of Child Nodes per Node:

Enter the maximum number of child nodes that an element in an XML message is allowed to have.

Maximum Number of Attributes per Node:

Enter the maximum number of attributes that an element is allowed to have.

Threatening Content

Overview

The **Threatening Content** filter can run a series of regular expressions that identify different attack signatures against request messages to see if they contain threatening content. Each expression identifies a particular attack signature, which can then be run against different parts of the request, including the request body, HTTP headers, and the request query string. Furthermore, it is possible to configure the MIME types on which the **Threatening Content** filter will operate.

The threatening content regular expressions are stored in the global **Blacklist** library, which can be viewed as a top level tree item on the tree view of the SOA Security Gateway Management Console. By default, the library contains regular expressions to identify SQL syntax to guard against SQL injection attacks, DOCTYPE DTD references to avoid against DTD expansion attacks, Java exception stack trace information to prevent call stack information getting returned to the client, and other expressions to identify many more types of attack signature.

The **Threatening Content** filter is available from the **Content Filtering** category of filters. Drag and drop the filter on to the circuit editor and enter a name for the filter in the **Name** field. The next sections describe how to configure the other tabs on this filter screen.

Scanning Details

Message Parts to Scan:

This section is used to configure what parts of the incoming request are to be scanned for threatening content. By default, the **Threatening Content** filter acts on the request body. However, it is also possible to scan the HTTP headers and the request query string for threatening content. Check the appropriate checkboxes to indicate what part of the request message you want to scan.

Blacklist:

The table lists all the regular expressions that have been added to the global **Blacklist**. These regular expressions are used to identify "threatening content". For example, there are regular expressions to match SQL syntax, ASCII control characters, and XML processing instructions, all of which can be used to attack a Web Service. Please refer to the Blacklist help guide for more information on how to configure the global **Blacklist**.

Select the regular expressions that you want to run against incoming requests using the

checkboxes in the table. New expressions can be added using the **Add** button. It is important to note that when adding new regular expressions on the **Add Regular Expression** dialog, the expressions are actually added to the global **Blacklist** library.

Existing regular expressions can be edited or removed by selecting the expression in the tree and selecting the **Edit** and **Delete** buttons respectively.

MIME Types

The **MIME Types** tab lists the MIME types that are to be scanned for in incoming messages. By default, all text- and XML-related types are scanned for threatening content. However, it is possible to select any type from the list.

Similar to the way in which the **Blacklist** regular expressions are global, so too are the MIME types. They can be added globally by right-clicking on the location of the Entity Store in the tree view in the SOA Security Gateway Management Console, and selecting the **MIME/DIME** menu option.

New types can be added by selecting the **Add** button and entering a type name and corresponding extension on the **Configure MIME/DIME Type** dialog. It is possible to enter a list of extensions by separating them with a spaces.

Existing types can be edited or deleted by selecting the **Edit** and **Delete** buttons.

Integrity XML-Signature Verification

Overview

In addition to validating XML Signatures for authentication purposes, the SOA Security Gateway can also use XML Signatures to prove message integrity. By signing an XML message, a client can be sure that any changes made to the message will not go un-noticed by the SOA Security Gateway. Therefore by validating the XML Signature on a message, the SOA Security Gateway can guarantee the *integrity* of the message.

Before configuring the **XML Signature Verification** filter, enter a name for this filter in the **Name** field.

Signature Location

Because there may be multiple signatures contained within the message, it is necessary to specify which signature the SOA Security Gateway should use to verify the integrity of the message. The signature can be extracted from one of three places:

- From the SOAP header
- Using WS-Security Actors
- Using XPath

Select the appropriate option from the dropdown.

What Must Be Signed

This section defines the content that must be signed in order for a SOAP message to pass the filter. This ensures that the client has signed something meaningful (i.e. part of the SOAP message) as opposed to some arbitrary data that would pass a "blind" signature validation. This further strengthens the integrity verification process.

An XPath expression is used to identify the nodeset that should be signed. To specify that nodeset, select either an existing XPath expression from the **XPath Expression** dropdown list, or add a new one using the **Add** button. XPath expressions can also be edited or removed with the **Edit** and **Delete** buttons respectively.

Signer's Public Key/Certificate

Select the **Certificate in Message** radio button in order to use the certificate from the

XML-Signature specified in the **Signature Location** section. The certificate will be extracted from the `KeyInfo` block.

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="User">
  ...
  <dsig:KeyInfo>
    <dsig:X509Data>
      <dsig:X509SubjectName>CN=User...</dsig:X509SubjectName>
      <dsig:X509Certificate>
        MIIE ..... EQgJ
      </dsig:X509Certificate>
    </dsig:X509Data>
    <dsig:KeyValue>
      <dsig:RSAKeyValue>
        <dsig:Modulus>
          AMfb2tT53GmMiD
          ...
          NmrNht7iy18=
        </dsig:Modulus>
        <dsig:Exponent>AQAB</dsig:Exponent>
       </dsig:RSAKeyValue>
    </dsig:KeyValue>
  </dsig:KeyInfo>
</dsig:Signature>
```

Clients may not always want to include their public keys in their signatures. In such cases, the public key can be retrieved from either a specified LDAP directory or from the **Certificate Store**.

To configure the SOA Security Gateway to retrieve a client certificate (and hence public key) from an LDAP directory, select the **Certificate in LDAP** radio button and then click on the **Add/Edit** button. A previously configured LDAP directory can be selected by choosing one from the **LDAP Source** dropdown list.

Alternatively, select the **Certificate in Store** radio button and then click the **Select** button to choose a certificate from the global Certificate Store. This certificate will then be associated with the incoming message, and will subsequently be used when validating the selected signature on the message.

For example, the following signed XML message does not include the signatory's certificate. Instead only the Common Name of the signatory's certificate is included. In this case, the SOA Security Gateway must obtain the certificate from either the LDAP directory or the **Certificate Store** in order to validate the signature on the message.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
<soap-env:Header>
```

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="User">
 <dsig:SignedInfo>
  <dsig:CanonicalizationMethod
       Algorithm="http://www.w3.org/2001/10/xml-exc-c14n"/>
  <dsig:SignatureMethod
       Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
  <dsig:Reference URI="">
   <dsig:Transforms>
    <dsig:Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
     <dsig:XPath>ancestor-or-self::soap-env:Body</dsig:XPath>
    </dsig:Transform>
    <dsig:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n"/>
   </dsig:Transforms>
   <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
   <dsig:DigestValue>rvJMkZ1RDo3pNfqCUBa4Qhs8i+M=</dsig:DigestValue>
  </dsig:Reference>
 </dsig:SignedInfo>
 <dsig:SignatureValue>
        AXL2gKhqqKwcKujVPftVoztySvtCdARGf97Cjt6Bbpf0w8QFiNuLJncQVnKB
        cQ+91KvudYZ/Sk8u7tXhoEiLvNwg76B2STPh+ypEWO+J7OSPedlUdnfVRRvW
        vjYLwJVjGNZ+mMTxvfO1wwcIb2Hg94n1BOaeBrNJ+2uO4i87W5TyufAGI+V8
        S6oSpPc5KQeHLXoyHS2+fXyqReSiwdhOeli4D4xT+HbjRgYJIwIikXn2k1Fr
        D/hnd1/xVf/LjrOwoY9id8W3IcZAzMIRh5SBZjWHYOQzk79xy4YDpzNVYIOB
        laAFqzg9G+Z4VYj+RdgrIVHhOXt+mq+fGZV6VheWGQ==
 </dsig:SignatureValue>
 <dsig:KeyInfo>
  <dsig:KeyName>
    CN=User,OU=R&amp;amp;D,O=Company Ltd.,L=Dublin 4,ST=Dublin,C=IE
  </dsig:KeyName>
 </dsig:KeyInfo>
</dsig:Signature>
</soap-env:Header>
<soap-env:Body>
    <ns1:getTime xmlns:ns1="urn:timeservice">
        </ns1:getTime>
 </soap-env:Body>
</soap-env:Envelope>
```

References

Digital Signatures [http://www.w3.org/TR/REC-DSig-label/]

XML Signature [http://www.w3.org/Signature/]

XML Signature Syntax and Processing [http://www.w3.org/TR/xmldsig-core/]

Introduction to Public-Key Cryptography [http://docs.sun.com/source/816-6154-10/]

Simple Object Access Protocol (SOAP) [http://www.w3.org/TR/SOAP/]

Sign Message

Overview

The SOA Security Gateway can sign both SOAP and non-SOAP XML messages. Attachments to the message can also be signed. The resultant XML signature is inserted into the message for consumption by a downstream Web Service. At the Web Service, the signature can be used to authenticate the message sender and/or verify the integrity of the message.

Enter a name for the filter in the **Name** field. The **Signature** and **Certificate** tabs can be used to configure how the signature is generated and where it is to be inserted in the downstream message.

1. Signature Tab

The **Signature** tab allows you to configure **What to sign** and **Where to place signature**. It is possible to use WSU IDs, XPath transforms, or general IDs to specify the nodeset that is to be signed. The generated Signature can be placed in the SOAP Header, inside a WS-Security block, or at a location pointed to by an XPath expression.

Section 1: What to Sign:

With WSU IDs, an ID attribute is inserted into the root element of the nodeset that is to be signed. The XML Signature then references this ID to indicate to verifiers of the Signature the nodes that were signed. The use of WSU IDs is the default option since they are WS-I compliant.

Alternatively, a generic ID attribute (that is not bound to the WSU namespace) can be used to dereference the data. The ID attribute is inserted into the top-level element of the nodeset that is to be signed. The generated XML Signature can then reference this ID to indicate what nodes were signed.

When XPath transforms are used, an XPath expression that points to the root node of the nodeset that is signed will be inserted into the XML Signature. When attempting to verify the Signature, this XPath expression must be run on the message to retrieve the signed content.

Use WSU IDs:

Select this option to reference the signed data using a `wsu:Id` attribute. In this case, a `wsu:Id` attribute is inserted into the root node of the nodeset that is signed. This id is then referenced in the generated XML Signature as an indication of what nodes were signed. The following example shows the correlation:

```
<soap:Envelope xmlns:soap="...">
  <soap:Header>
  <wsse:Security xmlns:wsse="...">
    <dsig:Signature xmlns:dsig="..." Id="Id-00000112e2c98df8-0000000000000004">
      <dsig:SignedInfo>
        <dsig:CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        <dsig:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <dsig:Reference URI="#Id-00000112e2c98df8-0000000000000003">
          <dsig:Transforms>
            <dsig:Transform
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          </dsig:Transforms>
          <dsig:DigestMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <dsig:DigestValue>xChPoiWJJrrPZkbXN8FPB8S4U7w=</dsig:DigestValue>
        </dsig:Reference>
      </dsig:SignedInfo>
      <dsig:SignatureValue>KG4N .... /9dw==</dsig:SignatureValue>
      <dsig:KeyInfo Id="Id-00000112e2c98df8-0000000000000005">
        <dsig:X509Data>
          <dsig:X509Certificate>
            MIID ... ZiBQ==
          </dsig:X509Certificate>
        </dsig:X509Data>
      </dsig:KeyInfo>
    </dsig:Signature>
  </wsse:Security>
  </soap:Header>
   <soap:Body xmlns:wsu="..." wsu:Id="Id-00000112e2c98df8-0000000000000003">
  <vs:getProductInfo xmlns:vs="http://ww.ca.com">
   <vs:Name>SOA Test Client</vs:Name>
   <vs:Version>5.0</vs:Version>
  </vs:getProductInfo>
 </s:Body>
</s:Envelope>
```

In the above example, a `wsu:Id` attribute has been inserted into the `<soap:Body>` element. This `wsu:Id` attribute is then referenced by the `URI` attribute of the `<dsig:Reference>` element in the actual Signature.

When the Signature is being verified, the value of the `URI` attribute can be used to locate the nodes that have been signed.

Once you have opted to use WSU IDs (by selecting the **Use WSU IDs** radio button), you can then either select the nodes to sign from a pre-configured list of typically signed nodesets, or use an XPath expression instead.

Select the **Locate Nodes** radio button and then select one or more nodesets to sign from the default list. By default it is possible to select the SOAP Body, WS-Security block, and SAML assertion. For each of these blocks, there are several namespace op-

tions available. For example, it is possible to sign both a SOAP 1.1 and/or a SOAP 1.2 block by distinguishing between their namespaces.

It is also possible to add more default nodesets by clicking on the **Add** button. The **Element Name**, **Namespace**, and **Index** of the nodeset can be entered in the field provided. The **Index** field is used to distinguish between 2 elements of the same name that occur within the same message.

Alternatively, if you do not wish to use any of the "standard" nodesets, it is possible to use an XPath expression instead. Select the **Use XPath** radio button (instead of the **Locate Nodes** button), and enter an XPath expression in the field provided.

Use XPath Transform:

Select this option if you wish to use an XPath transform to reference the signed content. An **XPath Predicate** must be entered in the field provided in order to do this.

For more information on how to configure XPath expressions, take a look at the Configuring XPath Expressions tutorial.

To illustrate the use of XPath predicates, we will show how the following SOAP message is signed when the default *Sign SOAP Body* predicate is selected:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Body>
  <vs:getProductInfo xmlns:vs="http://ww.ca.com">
   <vs:Name>SOA Test Client</vs:Name>
   <vs:Version>5.0</vs:Version>
  </vs:getProductInfo>
 </s:Body>
</s:Envelope>
```

The default XPath expression (i.e. "Sign SOAP Body") identifies the contents of the SOAP `Body` element, including the `Body` element itself. The following is the XML Signature produced when this XPath predicate is used:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
  <dsig:Signature id="Sample" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
   <dsig:SignedInfo>
     ...
    <dsig:Reference URI="">
     <dsig:Transforms>
      <dsig:Transform
```

```
          Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
        <dsig:XPath xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
          ancestor-or-self::soap:Body
        </dsig:XPath>
      </dsig:Transform>
      <dsig:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n"/>
     </dsig:Transforms>
       ...
    </dsig:Reference>
   </dsig:SignedInfo>
    ...
  </dsig:Signature>
 </s:Header>
 <s:Body>
  <vs:getProductInfo xmlns:vs="http://ww.ca.com">
   <vs:Name>SOA Test Client</vs:Name>
   <vs:Version>5.0</vs:Version>
  </vs:getProductInfo>
 </s:Body>
</s:Envelope>
```

The above XML Signature includes an extra `Transform` element, which has a child `XPath` element. This element specifies the XPath predicate that validating applications must use to identify the signed content.

Use IDs:

Select this option in order to use generic IDs (that are not bound to the WSU namespace) to dereference the signed data. Under this schema, the `URI` attribute of the `<Reference>` points at an ID attribute, which is inserted into the top-level node of the nodeset that is signed. Take a look at the following example, noting how the ID specified in the Signature matches the ID attribute that has been inserted into the `<Body>` element, indicating that the Signature applies to the entire contents of the SOAP Body.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
                Id="Id-0000011a101b167c-0000000000000013">
      <dsig:SignedInfo>
        <dsig:CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        <dsig:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <dsig:Reference URI="#Id-0000011a101b167c-0000000000000012">
          <dsig:Transforms>
            <dsig:Transform
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          </dsig:Transforms>
          <dsig:DigestMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
```

```
          <dsig:DigestValue>JCy0JoyhVZYzmrLrl92nxfr1+zQ=</dsig:DigestValue>
        </dsig:Reference>
      </dsig:SignedInfo>
      <dsig:SignatureValue>......<dsig:SignatureValue>
      <dsig:KeyInfo Id="Id-0000011a101b167c-0000000000000014">
        <dsig:X509Data>
          <dsig:X509Certificate>......</dsig:X509Certificate>
        </dsig:X509Data>
      </dsig:KeyInfo>
    </dsig:Signature>
  </soap:Header>
 <soap:Body Id="Id-0000011a101b167c-0000000000000012">
  <product version="5.0">
    <name>VordelSecure</name>
    <company>Vordel</company>
    <description>Web Services Security</description>
  </product>
 </soap:Body>
</soap:Envelope>
```

Section 2: Signing Key:

Signing Key:

Select the key to use to sign messages by clicking on the **Signing Key** button and selecting a certificate from the Certificate Store. The chosen certificate should contain a private key that can be used to sign the message. Alternatively, the certificate may have a corresponding private key stored in a Hardware Security Module (HSM). Details of the HSM can be specified on the **Configure Certificate and Private Key** dialog, which is used to edit certificates stored in the Certificate Store. Please refer to the Certificate Store help page for more information.

The *Distinguished Name* of the selected certificate will appear in the `X509SubjectName` element as follows:

```
<dsig:X509SubjectName>
   CN=Sample,OU=R&amp;amp;D,O=Company Ltd.,L=Dublin 4,ST=Dublin,C=IE
</dsig:X509SubjectName>
```

Section 3: Where to Place Signature:

Append Signature to Root or SOAP Header:

If the message is a SOAP message, the signature will be inserted into the SOAP `Header` element when this radio button is selected. The XML Signature will be inserted as an immediate child of the SOAP `Header` element. The following example shows a skeleton SOAP message which has been signed using this option:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
  <ws:Security xmlns:ws="http://schemas.xmlsoap.org/..." s:actor="test">
   <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/..." id="Sample">
    ...
   </dsig:Signature>
  </ws:Security>
 </s:Header>
 <s:Body>
  ...
 </s:Body>
</s:Envelope>
```

If, on the other hand, the message is just plain XML, the signature will be inserted as an immediate child of the root element of the XML message. The following example shows a non-SOAP XML message which has been signed using this option:

```
<PurchaseOrder>
 <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
  ...
 </dsig:Signature>

 <Items>
 ...
 </Items>
</PurchaseOrder>
```

Place in WS-Security Element for SOAP Actor/Role:

By selecting this option, the XML Signature will be inserted into the WS-Security element identified by the specified SOAP *actor* or *role*. A SOAP actor/role is simply a way of distinguishing a particular WS-Security block from others which may be present in the message. Actors belong to the SOAP 1.1 specification, but were replaced in SOAP 1.2 by roles. Conceptually, however, they are identical.

Enter the name of the SOAP actor or role of the WS-Security block in the dropdown. The following SOAP message contains an XML Signature within a WS-Security block identi-

fied by the "test" actor:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
  <ws:Security xmlns:ws="http://schemas.xmlsoap.org/..." s:actor="test">
   <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/..." id="Sample">
       ...
   </dsig:Signature>
  </ws:Security>
 </s:Header>
 <s:Body>
  ...
 </s:Body>
</s:Envelope>
```

Use XPath Location:

This option is useful in cases where the signature must be inserted into a non-SOAP XML message. In such cases, it is possible to insert the signature into a location pointed to by an XPath expression. Select or add an XPath expression in the field provided, and then specify whether the SOA Security Gateway should insert the signature *before* the location to which the XPath expression points, or *append* it to this location.

2. KeyInfo Tab

The fields on the **KeyInfo** tab allow you to configure how the signer's public key and certificate can be included (or not included) in the generated signature.

Do Not Include KeyInfo Section:

This option allows you to omit all information about the signatory's certificate from the signature. In other words, the `KeyInfo` element is omitted from the signature. This is useful where a downstream Web Service uses an alternative method of authenticating the signatory, and wishes to use the signature for the sole purpose of verifying the integrity of the message. In such cases, adding certificate information to the message may be regarded as an unnecessary overhead.

Include Certificate:

This is the default option which places the signatory's certificate inside the XML Signature itself. The following example, shows an example of an XML Signature which has been created using this option:

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
  ...
 <dsig:KeyInfo>
  <dsig:X509Data>
   <dsig:X509SubjectName>CN=Sample...</dsig:X509SubjectName>
   <dsig:X509Certificate>
       MIIEZDCCA0yg
       ....
       RNp9aKD1fEQgJ
   </dsig:X509Certificate>
  </dsig:X509Data>
 </dsig:KeyInfo>
</dsig:Signature>
```

Expand Public Key:

The details of the signatory's public key are inserted into a `KeyValue` block. The `KeyValue` block is only inserted when this option is checked.

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
   ...
 <dsig:KeyInfo>
  <dsig:X509Data>
   <dsig:X509SubjectName>CN=Sample...</dsig:X509SubjectName>
   <dsig:X509Certificate>
      MIIE ....... EQgJ
   </dsig:X509Certificate>
  </dsig:X509Data>
  <dsig:KeyValue>
   <dsig:RSAKeyValue>
    <dsig:Modulus>
       AMfb2tT53GmMiD
       ...
       NmrNht7iy18=
    </dsig:Modulus>
    <dsig:Exponent>AQAB</dsig:Exponent>
   </dsig:RSAKeyValue>
  </dsig:KeyValue>
 </dsig:KeyInfo>
</dsig:Signature>
```

Include Distinguished Name:

If this checkbox is checked, the Distinguished Name of the signatory's X.509 certificate

will be inserted in an <X509SubjectName> element as shown in the following example:

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
  ...
 <dsig:KeyInfo>
  <dsig:X509Data>
   <dsig:X509SubjectName>CN=Sample,C=IE...</dsig:X509SubjectName>
   <dsig:X509Certificate>
       MIIEZDCCA0yg
       ....
       RNp9aKD1fEQgJ
   </dsig:X509Certificate>
  </dsig:X509Data>
 </dsig:KeyInfo>
</dsig:Signature>
```

Include Key Name:

This option allows you insert a key identifier, or KeyName, to allow the recipient to identify the signatory. Enter an appropriate value for the KeyName in the **Value** field. Typical values include Distinguished Names (DName) from X.509 certificates, key IDs, or email addresses. Specify whether the specified value is a **Text value** of a **Distinguished name attribute** by checking the appropriate radio button.

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
   ...
 <dsig:KeyInfo>
  <dsig:KeyName>test@ca.com</dsig:KeyName>
 </dsig:KeyInfo>
</dsig:Signature>
```

Put Certificate in an Attachment:

The SOA Security Gateway supports SOAP messages with attachments. By selecting this option, you can save the signatory's certificate to the file specified in the input field. This file can then be sent along with the SOAP message as a SOAP attachment.

From previous examples, it is clear that the user's certificate is usually placed inside a KeyInfo element. However, in this example, the certificate is actually contained within an attachment, and not within the XML Signature itself. Clearly, we need a way to reference the certificate from the XML Signature, so that validating applications can process

the signature correctly. This is the role of the SecuriyTokenReference block.

The SecuriyTokenReference block provides a generic way for applications to retrieve security tokens in cases where these tokens are not contained within the SOAP message. The name of the security token is specified in the URI attribute of the Reference element.

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
   ...
 <dsig:KeyInfo>
  <wsse:SecurityTokenReference xmlns:wsse="http://schemas.xmlsoap.org/ws/...">
   <wsse:Reference URI="c:\myCertificate.txt"/>
  </wsse:SecurityTokenReference>
 </dsig:KeyInfo>
</dsig:Signature>
```

When the message is actually sent, the certificate attachment will be given a "Content-Id" corresponding to the URI attribute of the Reference element. The following example shows what the complete multipart MIME SOAP message looks like as it is sent over the wire. It should help illustrate how the Reference element actually refers to the "Content-ID" of the attachment:

```
POST /adoWebSvc.asmx HTTP/1.0
Content-Length: 3790
User-Agent: SOA Security Gateway
Accept-Language: en
Content-Type: multipart/related; type="text/xml";
              boundary="----=Multipart-SOAP-boundary"

------=Multipart-SOAP-boundary
Content-Id: soap-envelope
Content-Type: text/xml; charset="utf-8";
SOAPAction=getQuote

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
   ...
 <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
    ...
  <dsig:KeyInfo>
   <ws:SecurityTokenReference xmlns:ws="http://schemas.xmlsoap.org/ws/...">
    <ws:Reference URI="c:\myCertificate.txt"/>
   </ws:SecurityTokenReference>
  </dsig:KeyInfo>
 </dsig:Signature>
    ...
</s:Envelope>

------=Multipart-SOAP-boundary
Content-Id: c:\myCertificate.txt
Content-Type: text/plain; charset="US-ASCII"
```

```
MIIEZDCCA0ygAwIBAgIBAzANBgkqhki
....
7uFveG0eL0zBwZ5qwLRNp9aKD1fEQgJ
------=Multipart-SOAP-boundary-
```

Security Token Reference

A <wsse:SecurityTokenReference> element can be used to point to the security token used in the generation of the signature. Select this option if you wish to use this element. The type of the reference must be selected from the **Reference Type** dropdown.

The <wsse:SecurityTokenReference>, (within the <dsig:KeyInfo>), may contain a <wsse:Embedded> security token. Alternatively, the <wsse:SecurityTokenReference>, (within the <dsig:KeyInfo>), may refer to a certificate via a <dsig:X509Data>. Select the appropriate button, **Embed** or **Refer**, depending on whether you want to use an embedded security token or a referred one.

3. Advanced Tab

The **Advanced** tab allows you to configure how the signatory's certificate is to be included (if at all) in the XML Signature. It is also possible to specify any other sections of the XML message that are to be signed. The following 3 sections can be configured on this screen:

- Additional Elements to Sign
- Timestamp Options
- Advanced Options

Additional Elements to Sign:

The options here allow you to select other parts of the message that you may wish to sign.

- **Sign Certificate:**

  The certificate can be signed to prevent people cut-and-pasting other certificates into the message.

- **Sign Timestamp:**

  As stated earlier, timestamps are used to prevent replay attacks. However, to guarantee the end-to-end integrity of the timestamp, it is necessary to sign it. Note that this option is only enabled when the **Include timestamp in signature** option in the **Advanced Options** section is checked on the **Signature** tab.

- **Sign Attachments:**

  In addition to signing some or all of the contents of the SOAP message, it is also possible to sign attachments to the SOAP message. To sign all attachments, check the **Include Attachments** checkbox.

  A signed attachment is referenced in an XML Signature using the *Content-Id* or *cid* of the attachment. The `URI` attribute of the `Reference` element corresponds to this Content-Id. The following example shows how an XML Signature refers to a sample attachment. It shows the wire format of the message and its attachment as they are sent to the destination Web Service. Multiple attachments will result in successive `Reference` elements.

```
POST /myAttachments HTTP/1.0
Content-Length: 1000
User-Agent: SOA Security Gateway
Accept-Language: en
Content-Type: multipart/related; type="text/xml";
              boundary="----=Multipart-SOAP-boundary"

------=Multipart-SOAP-boundary
Content-Id: soap-envelope
SOAPAction: none
Content-Type: text/xml; charset="utf-8"

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
  <dsig:Signature id="Sample" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
   <dsig:SignedInfo>
    <dsig:CanonicalizationMethod
        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n"/>
    <dsig:SignatureMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <dsig:Reference URI="cid:moredata.txt">...</dsig:Reference>
   </dsig:SignedInfo>
  </dsig:Signature>
 </s:Header>
 <s:Body>
    ...
 </s:Body>
</s:Envelope>

------=Multipart-SOAP-boundary
Content-Id: moredata.txt
```

```
Content-Type: text/plain; charset="UTF-8"

Some more data.
------=Multipart-SOAP-boundary--
```

Timestamp Options:

It is possible to insert a timestamp into the message to indicate when exactly the signature was generated. Consumers of the signature can then validate the signature to ensure that it is not of date.

The following options are available:

1. **No Timestamp:**

   No timestamp will be inserted into the signature.

2. **Embed in WSSE Security:**

   The "wsu:Timestamp" will be inserted into a "wsse:Security" block. The Security block is identified by the SOAP actor/role specified on the **Signature** tab.

3. **Embed in Signature Property:**

   The "wsu:Timestamp" will be placed inside a signature property element in the "dsig:Signature".

The **Expires In** fields allow the user to optionally specify the "wsu:Expires" for the "wsu:Timestamp". If all fields are left at "0", no "wsu:Expires" element will be placed inside the "wsu:Timestamp".

The following examples shows a "wsu:Timestamp" that has been inserted into a "wsse:Security" block:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
   <wsse:Security>
     <wsu:Timestamp wsu:Id="Id-0000011294a0311e-000000000000003d">
      <wsu:Created>2007-05-16T11:22:45Z</wsu:Created>
      <wsu:Expires>2007-05-23T11:22:45Z</wsu:Expires>
```

```
    </wsu:Timestamp>
    <dsig:Signature ...>
    ...
    </dsig:Signature ...>
   </wsse:Security>
 </s:Header>
 <s:Body>
    ...
 </s:Body>
</s:Envelope>
```

Advanced Options:

This section allows you to configure very specific information about the signature generation process, including the signature method and key length to use, the digest algorithm, and whether to use an enveloped signature or not. The following sections describe each of these options in turn.

Add Inclusive Namespaces for Exclusive Canonicalization:

It is possible to include information about the namespaces (and their associated prefixes) of signed elements in the signature itself. This ensures that namespaces that are in the same scope as the signed element, but not directly or "visibly" used by this element, are included in the signature. This ensures that the signature can be validated as a standalone entity outside of the context of the message from which it was extracted.

It is also worth pointing out that the WS-I specification only permits the use of exclusive canonicalization in an XML Signature. The `<InclusiveNamespaces>` element is an attempt to take advantage of some of the behavior of *inclusive* canonicalization, while maintaining the simplicity of *exclusive canonicalization*.

A *PrefixList* attribute is used to list the prefixes of in-scope, but not visibly used elements and attributes. The following example shows how the `PrefixList` attribute is used in practice:

```
<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope'>
 <soap:Header>
  <wsse:Security xmlns:wsse='http://docs.oasis-open.org/...'
                 xmlns:wsu='http://docs.oasis-open.org/...'>
   <wsse:BinarySecurityToken wsu:Id='SomeCert'
                             ValueType="http://docs.oasis-open.org/...">
   lui+Jy4WYKGJW5xM3aHnLxOpGVIpzSg4V486hHFe7sH
   </wsse:BinarySecurityToken>
   <ds:Signature xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
    <ds:SignedInfo>
     <ds:CanonicalizationMethod
```

```
        Algorithm='http://www.w3.org/2001/10/xml-exc-c14n#'>
      <c14n:InclusiveNamespaces
          xmlns:c14n='http://www.w3.org/2001/10/xml-exc-c14n#'
          PrefixList='wsse wsu soap' />
     </ds:CanonicalizationMethod>
     <ds:SignatureMethod
          Algorithm='http://www.w3.org/2000/09/xmldsig#rsa-sha1'/>
     <ds:Reference URI=''>
      <ds:Transforms>
       <dsig:XPath xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
                   xmlns:m='http://example.org/ws'>
       //soap:Body/m:SomeElement
       </dsig:XPath>
       <ds:Transform Algorithm='http://www.w3.org/2001/10/xml-exc-c14n#'>
        <c14n:InclusiveNamespaces
              xmlns:c14n='http://www.w3.org/2001/10/xml-exc-c14n#'
              PrefixList='soap wsu test' />
       </ds:Transform>
      </ds:Transforms>
      <ds:DigestMethod Algorithm='http://www.w3.org/2000/09/xmldsig#sha1' />
      <ds:DigestValue>VEPKwzfPGOxh2OUpoK0bcl58jtU=</ds:DigestValue>
     </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>+diIuEyDpV7qxVoUOkb5rj61+Zs=</ds:SignatureValue>
    <ds:KeyInfo>
     <wsse:SecurityTokenReference>
      <wsse:Reference URI='#SomeCert' />
     </wsse:SecurityTokenReference>
    </ds:KeyInfo>
   </ds:Signature>
  </wsse:Security>
 </soap:Header>
 <soap:Body xmlns:wsu='http://docs.oasis-open.org/...'
            xmlns:test='http://www.test.com' wsu:Id='TheBody'>
  <m:SomeElement xmlns:m='http://example.org/ws' attr1='test:fdwfde' />
 </soap:Body>
</soap:Envelope>
```

Indent:

Select this method to ensure that the generated signature is properly indented.

Signature Method:

The **Signature Method** field allows you to configure the method used to generate the signature. The following 2 options are available:

- ***RSA-SHA1 Signature:***

  An RSA-SHA1 signature requires a public-private key pair. The private key gener-

ates the signature, i.e. signs the <dsig:SignedInfo>. The public key is required to verify the signature.

- ***HMAC-SHA1 Signature:***

  A HMAC-SHA1 signature also requires a public-private key pair. A random symmetric key is created and used to generate the signature, i.e. sign the <dsig:SignedInfo>. The public key is used to encrypt the random symmetric key at signature generation time. In order to verify the signature, the private key is used to decrypt the symmetric key. The symmetric key is then used to verify the signature.

HMAC Key Length:

This option allows the user to specify the length of the key to use when performing signatures of type "HMAC-SHA". A minimum key length of 20 bytes can be used, however the default length is 64 bytes.

It is important to note here that if you are using this filter in conjunction with an **XML Encryption Settings** filter and have elected to reuse the symmetric key by selecting the **Use Symmetric Key from Signature filter** option on the **Advanced** tab of the encryption filter, the symmetric key length must be appropriate for the **Encryption Algorithm** selected on the **Advanced tab** tab of the **XML-Encryption Settings** filter dialog.

The available encryption algorithms require key lengths as follows:

Encryption Algorithm	***Key Length (bytes)***
Triple-DES	192
AES-128	128
AES-192	192
AES-256	256

Use Derived Key:

A <wssc:DerivedKeyToken> token can be used to specify how derive a symmetric key from the original symmetric key held in <enc:EncryptedKey>. The derived symmetric key is used to generate the actual signature. It must be derived again during the verification process using the parameters in the <wssc:DerivedKeyToken>. One of these parameters is the decrypted symmetric key held in <enc:EncryptedKey>.

The following example shows the use of the

```
<enc:EncryptedKey Id="Id-0000010b8b0415dc-0000000000000000">
  <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
  <dsig:KeyInfo>
        ...
  </dsig:KeyInfo>
  <enc:CipherData>
</enc:EncryptedKey>

<wssc:DerivedKeyToken wsu:Id="Id-0000010bd2b8eca1-0000000000000017"
      Algorithm="http://schemas.xmlsoap.org/ws/2005/02/sc/dk/p_sha1">
  <wsse:SecurityTokenReference wsu:Id="Id-0000010bd2b8ed5d-0000000000000018">
    <wsse:Reference URI="#Id Id-0000010b8b0415dc-0000000000000000"
    ValueType="..../oasis-wss-soap-message-security-1.1#EncryptedKey"/>
  </wsse:SecurityTokenReference>
  <wssc:Generation>0</wssc:Generation>
  <wssc:Length>32</wssc:Length>
  <wssc:Label>WS-SecureConverstaionWS-SecureConverstaion</wssc:Label>
  <wssc:Nonce>h9TTWKRylCOz87+mc1/7Pg==</wssc:Nonce>
</wssc:DerivedKeyToken>

<dsig:Signature Id="Id-0000010b8b0415dc-0000000000000004">
  <dsig:SignedInfo>
    <dsig:CanonicalizationMethod
          Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <dsig:SignatureMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
    <dsig:Reference>...</dsig:Reference>
  </dsig:SignedInfo>
  <dsig:SignatureValue>...dsig:SignatureValue>
  <dsig:KeyInfo>
    <wsse:SecurityTokenReference wsu:Id="Id-0000010b8b0415dc-0000000000000006">
      <wsse:Reference
          URI="# Id-0000010bd2b8eca1-0000000000000017"
          ValueType="http://schemas.xmlsoap.org/ws/2005/02/sc/dk"/>
    </wsse:SecurityTokenReference>
  </dsig:KeyInfo>
</dsig:Signature>
```

Include Timestamp in Signature:

When this option is selected, a WS-Utility timestamp (i.e. `wsu:TimeStamp`) will be inserted into the XML Signature. The following skeleton signed SOAP message shows the timestamp:

```
<s:Header xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <ws:Security xmlns:ws="http://schemas.xmlsoap.org/ws/2002/04/secext"
     s:actor="test">
  <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
```

```
   <dsig:SignatureProperties Id="signatureProperties">
    <dsig:SignatureProperty Target="#Sample">
     <wsu:Timestamp wsu:Id="timestamp"
           xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility">
      <wsu:Created>2003.11.13T-09:36:39Z</wsu:Created>
     </wsu:Timestamp>
    </dsig:SignatureProperty>
   </dsig:SignatureProperties>
  </dsig:Signature>
 </ws:Security>
</s:Header>
```

Create Enveloped Signature:

By selecting this option, an enveloped XML Signature is generated. The following skeleton signed SOAP message shows the enveloped signature

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" id="Sample">
 <ds:SignedInfo>
  <ds:Reference URI="">
   <ds:Transforms>
    <ds:Transform
        Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
   </ds:Transforms>
  </ds:Reference>
 </ds:SignedInfo>
</ds:Signature>
```

This indicates to the application validating the signature that the signature itself should not be included in the signed data. In other words, to validate the signature, the application must first strip out the signature. This is necessary in cases where the entire SOAP envelope has been signed, and the resulting signature has been inserted into the SOAP header. In this case, the signature is over a nodeset which has been altered (i.e. the Signature has been inserted), and so the signature will break.

References

Digital Signatures [http://www.w3.org/TR/REC-DSig-label/]

XML Signature [http://www.w3.org/Signature/]

XML Signature Syntax and Processing [http://www.w3.org/TR/xmldsig-core/]

WS-Security [http://msdn.microsoft.com/ws/2002/04/Security]

WS-Utility [http://schemas.xmlsoap.org/ws/2002/07/utility]

Introduction to Public-Key Cryptography [http://docs.sun.com/source/816-6154-10/]

Simple Object Access Protocol (SOAP) [http://www.w3.org/TR/SOAP/]

Add HTTP Header

Overview

The SOA Security Gateway can add HTTP headers to a message as it passes through a policy. It is also possible to set a (base64-encoded) value for the header.

The **Add HTTP Header** filter can be used, for example, to add the message ID to a HTTP header. This message ID can then be forwarded to the destination Web Service, where messages can then be indexed and tracked by their IDs. In this way, it is possible to create a complete *audit trail* of the message from the time it is received by the SOA Security Gateway, until the time it is processed by the back-end system. The `${id}` wildcard, which represents the value of the unique message ID, can be used to add the message ID to the message.

Configuration

Follow these 3 steps to configure the **Add HTTP Header** filter:

1. Enter a name for this filter in the **Name** field.
2. Specify name of the HTTP header to add in the **HTTP Header Name** field.
3. Enter the value of the new HTTP header in the **HTTP Header Value** field. As stated earlier, it is possible to enter wildcards here to represent message attributes. At run-time, the SOA Security Gateway will replace these wildcards with the current value of the corresponding message attribute. For example, the `${id}` wildcard will be replaced by the value of the current message ID. Wildcards have the following syntax:

   ```
   ${messsage_attribute}
   ```

4. Check the **Base64 Encode** checkbox to configure the SOA Security Gateway to base64 encode the HTTP header value. This should be used, for example, if the header value is an X.509 certificate.

Remove Attachments

Overview

This filter can be used to remove ***all*** attachments from either a request or a response message, depending on where the filter is placed in the policy.

Configuration

Simply enter a name for this filter in the **Name** field.

Remove HTTP Header

Overview

The SOA Security Gateway can strip a named HTTP header from the message as it passes through a policy. This is especially useful in cases where end-user credentials are passed to the SOA Security Gateway in a HTTP header. After processing the credentials, the **Remove HTTP Header** filter can be used to strip the header from the message to ensure that it is not forwarded on to the destination Web Service.

Configuration

Follow these 3 steps to configure the **Remove HTTP Header** filter:

1. Enter a name for this filter in the **Name** field.
2. Specify name of the HTTP header to remove in the **HTTP Header Name** field.
3. Check the **Fail if header is not present** checkbox to configure the SOA Security Gateway to throw an exception if the message does not contain the named HTTP header.

Set Message

Overview

The **Set Message** filter replaces the body of the message. The replacement data can be plain text, HTML, XML, or any other text-based markup.

Wildcards representing the values of message attributes can be used in the replacement text to insert message-specific data into the message body. For example, it is possible to insert the authenticated user's ID into a `<Username>` element by using the `${authentication.subject.id}` wildcard as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Header>
  <Username>${authentication.subject.id}</Username>
 </soap:Header>
 <soap:Body>
  <getQuote xmlns="ca.com">
   <ticker>ORM.L</ticker>
  </getQuote>
 </soap:Body>
</soap:Envelope>
```

Assuming the user, "ca", authenticated successfully to the SOA Security Gateway, the message body will be set to the following contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Header>
  <Username>ca</Username>
 </soap:Header>
 <soap:Body>
  <getQuote xmlns="ca.com">
   <ticker>ORM.L</ticker>
  </getQuote>
 </soap:Body>
</soap:Envelope>
```

The **Set Message** filter can also be used to customize SOAP faults that are returned to

clients in the case of a failure or exception in the policy.

Configuration

Follow these 3 steps to configure the **Set Message** filter:

1. Enter a name for this filter in the **Name** field.
2. Specify the content type of the new message body in the **Content-type** field. For example, if the new message body is HTML markup, enter "text/html" in the **Content-Type** field.
3. Enter the new message body in the **Message Body** text area. Wildcards can be used, as illustrated in the example earlier, to ensure that current message attribute values are inserted into the message body at the appropriate places. It is also possible to load the message contents from a file by browsing to the location of the file using the **Browse** button.

Stylesheet Conversion

Overview

XSL (eXtensible Stylesheet Language) is an XML-based language which is used to create conversion stylesheets. An XSL stylesheet is used to transform an XML document into another document type. The stylesheet defines how elements in the XML source document should appear in the result document.

The SOA Security Gateway can convert XML data to other data formats using XSL files. For example, an incoming XML message adhering to one XML Schema can be converted to an XML message adhering to a different schema before it is sent to the destination Web Service.

These type of conversions are especially valuable in the Web Services arena, where a Web Service might receive SOAP requests from various types of clients, such as browsers, applications, PDAs, and WAP-enabled mobile phones. Each client might send up a different type of SOAP request to the Web Service. Using stylesheets, the Service can then convert each type of request to the same format. The requests can then be processed in the same fashion.

Configuration

Open the **Stylesheet Conversion** screen on the **SOA Security Gateway Management Console**.

Follow these 3 steps to configure the **Stylesheet Conversion** filter:

1. Enter a name for this conversion in the **Filter Name** field.
2. As stated earlier, it is possible to convert an incoming XML message to other data formats. In such cases the **Content-Type** field should be updated accordingly. So, for example, if the stylesheet converts an XML message to a HTML before routing it onwards, the **Content-Type** should be changed to "text/html;".
3. Select an XSL stylesheet from the **Stylesheet Location** dropdown, which is populated with the contents of the Stylesheet Library. A new stylesheet can be imported into the library by clicking on the **Import** button and then the **Add** button on the **Stylesheet Library** dialog.

Stylesheet Parameters

An XSL stylesheet can be parameterized by certain values using `<xsl:param>` elements.

These values can then be used in the templates defined throughout the stylesheet.

Using the **Stylesheet Conversion** filter, it is possible to pass the values of message attributes to the configured stylesheet. For example, it is possible to take the value of the `authentication.subject.id` message attribute, pass it to the configured XSL stylesheet, and then output this value to the result produced by the conversion.

To use this useful feature, select the checkbox and then specify the message attribute to pass to the stylesheet by clicking on the **Add** button.

As an example of how to configure this, take a look at the following excerpt from an XSL stylesheet that uses parameters:

```
<xsl:param name="authentication.subject.id"/>
<xsl:param name="authentication.issuer.id"/>
```

To pass the corresponding message attribute values to the stylesheet, it is necessary to add the `authentication.subject.id` and `authentication.issuer.id` message attributes to the **Message Attributes to use** table.

References

XSLT W3C Recommendation [http://www.w3.org/TR/xslt]

Relative Path

Overview

The **Relative Path** filter provides a way to identify an incoming XML message based on the *relative path* on which the message was received.

Configuration

The following example illustrates how to find the *relative path* of an incoming message. Consider the following SOAP message:

```
POST /services/helloService HTTP/1.1
Host: localhost:8095
Content-Length: 196
SOAPAction: HelloService
Accept-Language: en-US
UserAgent: SOA Security Gateway
Content-Type: text/XML; utf-8

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <getHello xmlns="http://www.ca.com/"/>
  </soap:Body>
</soap:Envelope>
```

The *relative path* for this message is as follows:

Relative Path
/services/helloService

To configure the **Relative Path** filter, complete the following:

- Enter a name for the filter in the **Name** field.
- Enter a regular expression to match the value of the relative path on which messages will be received in the **Relative Path** field. For example, enter "^/services$"

to exactly match a path with the value "/services". Incoming messages received on a matching relative path value will be passed on to the next filter on the success path in the policy.

SOAPAction

Overview

The **SOAPAction** filter provides a way to identify an incoming XML message based on the *SOAPAction* HTTP header in the message.

Configuration

The following example illustrates how to locate the *SOAPAction* header in an incoming message. Consider the following SOAP message:

```
POST /services/helloService HTTP/1.1
Host: localhost:8095
Content-Length: 196
SOAPAction: HelloService
Accept-Language: en-US
UserAgent: SOA Security Gateway
Content-Type: text/XML; utf-8

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <getHello xmlns="http://www.ca.com/"/>
  </soap:Body>
</soap:Envelope>
```

The *SOAPAction* for this message is as follows:

SOAPAction:
HelloService

To configure the **SOAPAction** filter, complete the following:

- Enter a name for the filter in the **Name** field.
- Enter a regular expression to match the value of the SOAPAction HTTP header in the **SOAPAction** field. For example, enter "^getQuote$" to exactly match a SOAPAction

header with the value "getQuote". Incoming messages with a matching SOAPAction value will be passed on to the next filter on the success path in the policy.

SOAP Operation

Overview

The **SOAP Operation** filter provides a way to identify an incoming XML message based on the *SOAP Operation* in the SOAP Body.

Configuration

The following example demonstrates how to find the *SOAP Operation* of an incoming message. Consider the following SOAP message:

```
POST /services/timeservice HTTP/1.0
Host: localhost:8095
Content-Length: 374
SOAPAction: TimeService
Accept-Language: en-US
UserAgent: SOA Security Gateway
Content-Type: text/XML; utf-8

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns1:getTime xmlns:ns1="urn:timeservice">
      <ns1:city>Dublin</ns1:city>
    </ns1:getTime>
  </soap:Body>
</soap:Envelope>
```

The *SOAP Operation* for this message and its namespace are as follows:

SOAP Operation:	getTime
SOAP Operation Namespace:	urn:timeservice

As can be seen from the example, the *SOAP Operation* is derived from the first element beneath the SOAP `<Body>` element.

To configure the **SOAP Operation** filter, complete the following:

- Enter a name for the filter in the **Name** field.

- Enter the name of the *SOAP Operation* in the **Operation** field. Incoming messages with an operation name matching the value entered here will be passed on to the next "Success" filter in the policy.
- Enter the namespace to which the *SOAP Operation* belongs in the **Namespace** field.

XML-Decryption

Overview

The **XML Decryption** filter is responsible for decrypting data within XML messages based on the settings configured in the **XML-Decryption Settings** filter.

The **XML-Decryption Settings** filter generates the `decryption.properties` message attribute based on its configuration settings. The **XML-Decryption** filter uses (or requires) these properties to perform the actual decryption of the data.

Configuration

Enter a name for the filter in the **Name** field.

Auto-generation using the XML Decryption Wizard

Since the **XML Decryption** filter must always be paired with an **XML Decryption Settings** filter, it makes sense to have a wizard that can generate both of these filters at the same time. To use the wizard, right-click on the name of the policy in the tree view of the SOA Security Gateway Management Console and select the **XML Decryption Settings** menu option.

Configure the fields on the **XML Decryption Settings** dialog as explained in the XML Decryption Settings help page. When finished, an **XML Decryption Settings** filter will be created along with an **XML Decryption** filter.

XML-Decryption Settings

Overview

The SOA Security Gateway can decrypt an XML encrypted message on behalf of its intended recipients. XML Encryption is a W3C standard that allows data to be encrypted and decrypted at the application layer of the OSI stack, thus ensuring complete end-to-end confidentiality of data.

The **XML-Decryption Settings** should be used in conjunction with the **XML-Decryption** filter, which actually performs the decryption. The **XML-Decryption Settings** generates the `decryption.properties` message attribute, which is required by the **XML-Decryption** filter.

It is important to note that the output of a successfully executed decryption filter is the original unencrypted message. Depending on whether or not the **Remove EncryptedKey used in decryption** has been enabled, all information relating to the encryption key can be removed from the message. Take a look at the Options section below for more information on this setting.

XML Encryption Overview

XML Encryption facilitates the secure transmission of XML documents between two application end-points. Whereas traditional transport-level encryption schemes, such as SSL and TLS, can only offer point-to-point security, XML Encryption guarantees complete end-to-end security. Encryption takes place at the application-layer and so the encrypted data can be encapsulated within the message itself. The encrypted data can therefore remain encrypted as it travels along its path to the target Web Service. Furthermore, the data is encrypted such that only its intended recipients can decrypt it.

In order to fully understand how the SOA Security Gateway decrypts XML encrypted messages, it is first necessary to take a look at the format of an XML Encryption block. The following example shows a SOAP message containing information about CA:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Body>
  <getCompanyInfo xmlns="www.ca.com">
   <name>Company</name>
   <description>XML Security Company</description>
  </getCompanyInfo>
 </s:Body>
</s:Envelope>
```

After encrypting the SOAP Body, the message appears as follows:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
  <Security xmlns="http://schemas.xmlsoap.org/ws/2003/06/secext" s:actor="Enc">
   <!-- Encapsulates the recipient's key details -->
   <enc:EncryptedKey xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
           Id="00004190E5D1-7529AA14" MimeType="text/xml">
    <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04xmlenc#rsa-1_5">
     <enc:KeySize>256</enc:KeySize>
    </enc:EncryptionMethod>
    <enc:CipherData>
     <!-- The session key encrypted with the recipient's public key -->
     <enc:CipherValue>
         AAAAAJ/lK ... mrTF8Egg==
     </enc:CipherValue>
    </enc:CipherData>
    <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
     <dsig:KeyName>sample</dsig:KeyName>
     <dsig:X509Data>
      <!-- The recipient's X.509 certificate -->
      <dsig:X509Certificate>
          MIIEZzCCA0 ... fzmc/YR5gA
      </dsig:X509Certificate>
     </dsig:X509Data>
    </dsig:KeyInfo>
    <enc:CarriedKeyName>Session key</enc:CarriedKeyName>
    <enc:ReferenceList>
     <enc:DataReference URI="#00004190E5D1-5F889C11"/>
    </enc:ReferenceList>
   </enc:EncryptedKey>
  </Security>
 </s:Header>
 <enc:EncryptedData xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
         Id="00004190E5D1-5F889C11" MimeType="text/xml"
         Type="http://www.w3.org/2001/04/xmlenc#Element">
  <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04xmlenc#aes256-cbc">
   <enc:KeySize>256</enc:KeySize>
  </enc:EncryptionMethod>
  <enc:CipherData>
   <!-- The SOAP Body encrypted with the session key -->
   <enc:CipherValue>
       E2ioF8ib2r ... KJAnrX0GQV
   </enc:CipherValue>
  </enc:CipherData>
  <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
   <dsig:KeyName>Session key</dsig:KeyName>
  </dsig:KeyInfo>
 </enc:EncryptedData>
<s:Envelope>
```

The following are the most important elements:

- `EncryptedKey`: The `EncryptedKey` element encapsulates all information relevant to

the encryption key.

- `Reference`: This element corresponds to a single message-digest, which was computed over a part of the message. As was stated earlier, one of the advantages of XML Signatures is that multiple users can sign different parts of the document. Therefore, there could be multiple `Reference` elements, each one pointing to a specific part of the message with the `URI` attribute.
- `EncryptionMethod`: The `Algorithm` attribute specifies the algorithm that was used to encrypt the data. It is worth noting that the message data (i.e. `EncryptedData`) is encrypted using the AES (Advanced Encryption Standard) ***symmetric cipher***, but the session key (i.e. `EncryptedKey`) is encrypted with the RSA ***asymmetric*** algorithm.
- `CipherValue`: The value of the encrypted data. The contents of the `CipherValue` element are always base64 encoded.
- `DigestValue`: Contains the base64-encoded message-digest.
- `KeyInfo`: Contains information about the recipient and his encryption key, such as the key name, X.509 certificate, and Common Name.
- `EncryptedData`: The actual XML element(s) or content that has been encrypted. In this case, the SOAP `Body` element has been encrypted, and so the `EncryptedData` block has replaced the SOAP `Body` element.

Now that we've seen how encrypted data can be encapsulated within an XML message, it is important to discuss how this data gets encrypted in the first place. Once we understand how data is encrypted, the fields that must be configured to decrypt this data will become easier to understand.

When a message is encrypted, it is encrypted in such a manner that only the intended recipient(s) of the message can decrypt it. By encrypting the message with the recipient's public key, the sender can be guaranteed that only the intended recipient can decrypt the message through the use of his private key, to which he has sole access. This is the basic principle behind *asymmetric cryptography*.

In practice, however, encrypting and decrypting data with a public-private keypair is a notoriously CPU-intensive and time consuming affair. Because of this, asymmetric cryptography is seldom used to encrypt large amounts of data. The following steps exemplify a more typical encryption process:

1. The sender generates a one-time *symmetric* (or session) key which will be used to encrypt the data. Symmetric key encryption is much faster than asymmetric encryption and is far more efficient when large amounts of data are involved.

2. The sender encrypts the data with the symmetric key. This same key can then be used to decrypt the data. It is therefore crucial that only the intended recipient can access the symmetric key and consequently decrypt the data.

3. To make sure that nobody else can decrypt the data, the symmetric key is encrypted with the recipient's *public key*.

4. The data (encrypted with the symmetric key) and session key (encrypted with the recipient's public key) are then sent together to the intended recipient.

5. When the recipient receives the message he decrypts the encrypted session key using his *private key*. Since the recipient is the only one with access to the private key, he is the only one who can decrypt the encrypted session key.

6. Armed with the decrypted session key, the recipient can simply decrypt the encrypted data into its original plaintext form.

Now that we understand how XML Encryption works, it is now time to learn how to configure the SOA Security Gateway to decrypt XML encrypted messages. The following sections describe how to configure the **XML Decryption Settings** filter to decrypt encrypted XML data.

Node(s) to Decrypt

An XML message may contain several `EncryptedData` blocks. This section allows the administrator to specify which encryption blocks are to be decrypted. There are two options available here:

1. Decrypt All Encrypted Nodes
2. Use XPath to Select Encrypted Nodes

Decrypt All Encrypted Nodes:

The SOA Security Gateway will attempt to decrypt ***all*** `EncryptedData` blocks contained within the message.

Use XPath to Select Encrypted Nodes:

This option allows the administrator to explicitly choose the `EncryptedData` block that the SOA Security Gateway should decrypt.

The following skeleton SOAP message contains two `EncryptedData` blocks:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
    ...
 <s:Header>
 <s:Body>
  <!-- 1st EncryptedData block -->
  <e:EncryptedData xmlns:e="http://www.w3.org/2001/04/xmlenc#"
          Encoding="iso-8859-1" Id="ENC_1" MimeType="text/xml"
          Type="http://www.w3.org/2001/04/xmlenc#Element">
     ...
  </e:EncryptedData>
  <!-- 2nd EncryptedData block -->
  <e:EncryptedData xmlns:e="http://www.w3.org/2001/04/xmlenc#"
          Encoding="iso-8859-1" Id="ENC_2" MimeType="text/xml"
          Type="http://www.w3.org/2001/04/xmlenc#Element">
     ...
  </e:EncryptedData>
 </s:Body>
</s:Envelope>
```

The `EncryptedData` blocks are selected using XPath. The following XPath expressions can be used to select the respective `EncryptedData` blocks:

EncryptedData Block	***XPath Expression***
1st	//enc:EncryptedData[@Id='ENC_1']
2st	//enc:EncryptedData[@Id='ENC_2']

Simply click the **Add**, **Edit**, or **Delete** buttons to add, edit, or remove an XPath expression respectively.

Check the **Fail if no encrypted data found** checkbox if the SOA Security Gateway should fail if it cannot find `EncryptedData` blocks in the message.

In order to remove the EncryptedKey element from the message after it has been used in the decryption operation, check the **Remove the EncryptedKey element used in decryption** checkbox.

Decryption Key

This section allows you to specify what key to use to decrypt the encrypted nodes. As discussed in the Overview section above, data encrypted with a public key can only be decrypted with the corresponding private key. In this section, you can elect to take the private (i.e. decryption) key from the `<KeyInfo>` element of the XML Encryption block

or else, the certificate stored in a CA message attribute can be used to lookup the private key of the intended recipient of the encrypted data in the Certificate Store.

Find via KeyInfo in Message:

Select this option if you want to extract the decryption key from the XML Encryption block in the message. The client that encrypted the message may or may not have added its private key (to use to decrypt the data that was encrypted with its public key) to the XML Encryption block within the message. So this option should only be used in cases where the private key has been included in the `<KeyInfo>` element within the XML Encryption block in the message.

Find via Certificate in Attribute:

In cases where the client has opted not to include its private key in the XML Encryption block, the key must be extracted from an alternative source so that the encrypted data can be decrypted.

Typically, a **Find Certificate** filter would be used in a policy to locate an appropriate certificate and store it in the `certificate` message attribute. Once the certificate has been stored in this attribute, the `XML Decryption Settings` filter can use this certificate to lookup the Certificate Store for a corresponding private key for the public key stored in the certificate. To do this, simply select the `certificate` attribute from the dropdown.

Options

The following configuration options are available in this section:

Fail if no encrypted data found:

If this option is selected, the filter will fail if no `<EncryptedData>` elements are found within the message.

Remove the EncryptedKey used in decryption:

Select this option to remove information relating to the decryption key from the message. By checking this option, the `<EncryptedKey>` block will be removed from the message.

It is important to note that in cases where the `<EncryptedKey>` block has been included within the `<EncryptedData>` block, it will be removed regardless of whether this setting has been checked or not.

Auto-generation using the XML Decryption Wizard

Since the **XML Decryption Settings** filter must always be paired with an **XML Decryption** filter, it makes sense to have a wizard that can generate both of these filters at the same time. To use the wizard, right-click on the name of the policy in the tree view of the SOA Security Gateway Management Console and select the **XML Decryption Settings** menu option.

Configure the fields on the **XML Decryption Settings** dialog as explained above. When finished, an **XML Decryption Settings** filter will be created along with an **XML Decryption** filter.

References

XML Encryption Syntax and Processing [http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/]

XML Encryption Requirements [http://www.w3.org/TR/xml-encryption-req]

XML Signature [http://www.w3.org/Signature/]

XML Signature Syntax and Processing [http://www.w3.org/TR/xmldsig-core/]

XPath W3C Recommendation [http://www.w3.org/TR/xpath]

Introduction to Public-Key Cryptography [http://docs.sun.com/source/816-6154-10/]

XML-Encryption

Overview

The **XML-Encryption** is responsible for encrypting (parts of) XML messages based on the settings configured in the **XML-Encryption Settings** filter.

The **XML-Encryption Settings** filter generates the `encryption.properties` message attribute based on its configuration settings. The **XML-Encryption** filter uses (or requires) these properties to perform the actual encryption of the data.

Configuration

Enter a name for the filter in the **Name** field.

Auto-generation using the XML Encryption Settings Wizard

Since the **XML Encryption** filter must always be used in conjunction with the **XML Encryption Settings** and **Find Certificate** filters, it makes sense to have a wizard that can generate this triplet of filters at the same time. To use this wizard, right-click on the name of the policy in the tree view of the SOA Security Gateway Management Console and select the **XML Encryption Settings** menu option.

For more information on how to configure the **XML Encryption Settings Wizard** please refer to the XML Encryption Settings Wizard help page.

XML-Encryption Settings

Overview

The SOA Security Gateway can XML encrypt an ingress XML message so that only certain specified recipients can decrypt the message. XML Encryption is a W3C standard that allows data to be encrypted and decrypted at the application layer of the OSI stack, thus ensuring complete end-to-end confidentiality of data.

The **XML-Encryption Settings** should be used in conjunction with the **XML-Encryption** filter, which actually performs the encryption. The **XML-Encryption Settings** generates the `encryption.properties` message attribute, which is required by the **XML-Encryption** filter.

XML Encryption Overview

XML Encryption facilitates the secure transmission of XML documents between two application end-points. Whereas traditional transport-level encryption schemes, such as SSL and TLS, can only offer point-to-point security, XML Encryption guarantees complete end-to-end security. Encryption takes place at the application-layer and so the encrypted data can be encapsulated within the message itself. The encrypted data can therefore remain encrypted as it travels along its path to the target Web Service.

Before explaining how to configure the SOA Security Gateway to encrypt XML messages, it is useful to take a look at what an XML encrypted message looks like. The following example shows a SOAP message containing information about CA:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Body>
  <getCompanyInfo xmlns="http://www.ca.com">
   <name>Company</name>
   <description>XML Security Company</description>
  </getCompanyInfo>
 </s:Body>
</s:Envelope>
```

After encrypting the SOAP Body, the message appears as follows:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
```

```
<s:Header>
 <Security xmlns="http://schemas.xmlsoap.org/ws/2003/06/secext" s:actor="Enc">
  <!-- Encapsulates the recipient's key details -->
  <enc:EncryptedKey xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
          Id="00004190E5D1-7529AA14" MimeType="text/xml">
   <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04xmlenc#rsa-1_5">
    <enc:KeySize>256</enc:KeySize>
   </enc:EncryptionMethod>
   <enc:CipherData>
    <!-- The session key encrypted with the recipient's public key -->
    <enc:CipherValue>
        AAAAAJ/lK ... mrTF8Egg==
    </enc:CipherValue>
   </enc:CipherData>
   <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
    <dsig:KeyName>sample</dsig:KeyName>
    <dsig:X509Data>
     <!-- The recipient's X.509 certificate -->
     <dsig:X509Certificate>
         MIIEZzCCA0 ... fzmc/YR5gA
     </dsig:X509Certificate>
    </dsig:X509Data>
   </dsig:KeyInfo>
   <enc:CarriedKeyName>Session key</enc:CarriedKeyName>
   <enc:ReferenceList>
    <enc:DataReference URI="#00004190E5D1-5F889C11"/>
   </enc:ReferenceList>
  </enc:EncryptedKey>
 </Security>
</s:Header>
<enc:EncryptedData xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
        Id="00004190E5D1-5F889C11" MimeType="text/xml"
        Type="http://www.w3.org/2001/04/xmlenc#Element">
 <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04xmlenc#aes256-cbc">
  <enc:KeySize>256</enc:KeySize>
 </enc:EncryptionMethod>
 <enc:CipherData>
  <!-- The SOAP Body encrypted with the session key -->
  <enc:CipherValue>
      E2ioF8ib2r ... KJAnrX0GQV
  </enc:CipherValue>
 </enc:CipherData>
 <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
  <dsig:KeyName>Session key</dsig:KeyName>
 </dsig:KeyInfo>
</enc:EncryptedData>
<s:Envelope>
```

The following are the most important elements:

- `EncryptedKey`:

  The `EncryptedKey` element encapsulates all information relevant to the encryption key.

- `Reference`:

  This element corresponds to a single message-digest, which was computed over a part of the message. As was stated earlier, one of the advantages of XML Signatures is that multiple users can sign different parts of the document. Therefore, there could be multiple `Reference` elements, each one pointing to a specific part of the message with the `URI` attribute.

- `EncryptionMethod`:

  The `Algorithm` attribute specifies the algorithm that was used to encrypt the data. It is worth noting that the message data (i.e. `EncryptedData`) is encrypted using the AES (Advanced Encryption Standard) ***symmetric cipher***, but the session key (i.e. `EncryptedKey`) is encrypted with the RSA ***asymmetric*** algorithm.

- `CipherValue`:

  The value of the encrypted data. The contents of the `CipherValue` element are always base64 encoded.

- `DigestValue`:

  Contains the base64-encoded message-digest.

- `KeyInfo`:

  Contains information about the recipient and his encryption key, such as the key name, X.509 certificate, and Common Name.

- `EncryptedData`:

  The actual XML element(s) or content that has been encrypted. In this case, the SOAP `Body` element has been encrypted, and so the `EncryptedData` block has replaced the SOAP `Body` element.

Now that we've seen how encrypted data can be encapsulated within an XML message, it is important to discuss how this data gets encrypted in the first place. (**Note:** the reader should refer to the References section for a more thorough explanation of the mechanics of public-key cryptography.)

When a message is encrypted, it is encrypted in such a manner that only the intended recipient(s) of the message can decrypt it. By encrypting the message with the recipient's public key, the sender can be guaranteed that only the intended recipient can decrypt the message through the use of his private key, to which he has sole access. This

is the basic principle behind *asymmetric cryptography*.

In practice, however, encrypting and decrypting data with a public-private keypair is a notoriously CPU-intensive and time consuming affair. Because of this, asymmetric cryptography is seldom used to encrypt large amounts of data. The following steps exemplify a more typical encryption process:

1. The sender generates a one-time *symmetric* (or session) key which will be used to encrypt the data. Symmetric key encryption is much faster than asymmetric encryption and is far more efficient when large amounts of data are involved.
2. The sender encrypts the data with the symmetric key. This same key can then be used to decrypt the data. It is therefore crucial that only the intended recipient can access the symmetric key and consequently decrypt the data.
3. To make sure that nobody else can decrypt the data, the symmetric key is encrypted with the recipient's *public key*.
4. The data (encrypted with the symmetric key) and session key (encrypted with the recipient's public key) are then sent together to the intended recipient.
5. When the recipient receives the message he decrypts the encrypted session key using his *private key*. Since the recipient is the only one with access to the private key, he is the only one who can decrypt the encrypted session key.
6. Armed with the decrypted session key, the recipient can simply decrypt the encrypted data into its original plaintext form.

Now that we understand the structure and mechanics of XML Encryption, it is time to learn how to configure the SOA Security Gateway to encrypt egress XML messages. The following chapters describe how to configure each of the sections on the **XML Encryption** screen.

Node(s) to Encrypt

This section allows the administrator to specify what part of the XML message is to be encrypted. An XPath expression is used to identify the nodeset that is to be signed.

To add a new XPath expression, select the **Add** button. Similarly, to edit or remove an existing expression, select the **Edit** and **Remove** buttons respectively.

Specify whether to encrypt the node or encrypt the node contents using either the **Encrypt Node** or the **Encrypt Node Contents** radio button.

Recipients

XML Messages can be encrypted for multiple recipients. In such cases, the symmetric session key is encrypted with the public key of each intended recipient and added to the message.

The following SOAP message has been encrypted for 2 recipients - *ca_1* and *ca_2*. The session key has been encrypted twice: once for *ca_1* using his public key, and a second time for *ca_2* using his public key. It is important to note that the data itself is only encrypted once, while the session key must be encrypted for each recipient. For illustration purposes, only those elements relevant to the above discussion have been included in the following XML encrypted message.

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
  <Security xmlns="http://schemas.xmlsoap.org/ws/2003/06/secext"
      s:actor="Enc Keys">
   <enc:EncryptedKey xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
       Id="0000418BBB61-A692675C" MimeType="text/xml">
    ...
    <enc:CipherData>
     <!-- Session key encrypted with sample's public key and base64-encoded -->
     <enc:CipherValue>AAAAAExx1A ... vuAhCgMQ==</enc:CipherValue>
    </enc:CipherData>
    <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
     <dsig:KeyName>ca_1</dsig:KeyName>
    </dsig:KeyInfo>
    <enc:CarriedKeyName>Session key</enc:CarriedKeyName>
     <enc:ReferenceList>
         <enc:DataReference URI="#0000418BBB61-D4495D9B"/>
    </enc:ReferenceList>
   </enc:EncryptedKey>
   <enc:EncryptedKey xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
       Id="#0000418BBB61-D4495D9B" MimeType="text/xml">
    ...
    <enc:CipherData>
     <!-- Session key encrypted with ca's public key and base64-encoded -->
     <enc:CipherValue>AAAAABZH+U ... MrMEEM/Ps=</enc:CipherValue>
    </enc:CipherData>
    <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
     <dsig:KeyName>ca_2</dsig:KeyName>
    </dsig:KeyInfo>
    <enc:CarriedKeyName>Session key</enc:CarriedKeyName>
    <enc:ReferenceList>
         <enc:DataReference URI="#0000418BBB61-D4495D9B"/>
    </enc:ReferenceList>
   </enc:EncryptedKey>
  </Security>
 </s:Header>
 <enc:EncryptedData xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
      Id="0000418BBB61-D4495D9B" MimeType="text/xml"
      Type="http://www.w3.org/2001/04/xmlenc#Element">
  <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04xmlenc#aes256-cbc">
   <enc:KeySize>256</enc:KeySize>
  </enc:EncryptionMethod>
  <enc:CipherData>
   <!-- SOAP Body encrypted with symmetric session key and base64-encoded -->
   <enc:CipherValue>WD0TmuMk9 ... GzYFeq8SM=</enc:CipherValue>
  </enc:CipherData>
  <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
```

```
   <dsig:KeyName>Session key</dsig:KeyName>
  </dsig:KeyInfo>
 </enc:EncryptedData>
</s:Envelope>
```

There are 2 `EncryptedKey` elements, one for each recipient. The `CipherValue` element contains the session key encrypted with the recipient's public key. The encrypted session key must be base64-encoded so that it can be represented as the textual contents of an XML element.

The `EncryptedData` element contains the actual encrypted data, along with information about the encryption process, including the encryption algorithm used, the size of the encryption key, and the type of data that was encrypted, for example, whether an element or the contents of an element was encrypted.

With this piece of background knowledge, the user should now be more familiar with the concept of *recipients* for an XML encrypted message. The **Recipient Group** field in the **Recipients** section allows the administrator to organize multiple recipients into **Recipient Groups**.

If the SOA Security Gateway fails to encrypt the message for any of the recipients in the selected **Recipient Group**, the filter will fail. Otherwise, the newly encrypted message will be routed onwards to its configured endpoint.

Configuring Recipients

To configure recipients, click the **Add** button on the **Recipients** tab.

Enter a friendly name for the recipient in the **Recipient Name** field. This name will appear in the table on the main **XML Encryption Settings** screen.

The SOA Security Gateway will insert the encrypted key into the WS-Security block specified in the **Actor** field.

Basic Tab:

On the **Basic** tab, select the message attribute that contains the public key to encrypt the data with. Only the recipient will be able to decrypt the data using the corresponding private key. The `certificate` is the message attribute that is most likely to contain the certificate to use, and so this is selected by default.

KeyInfo Tab:

The fields on the **KeyInfo** tab allow you to configure how the signer's public key and

certificate can be included (or not included) in the generated signature. The following fields are available:

Do Not Include KeyInfo Section:

This option allows you to omit all information about the signatory's certificate from the signature. In other words, the `KeyInfo` element is omitted from the signature. This is useful where a downstream Web Service uses an alternative method of authenticating the signatory, and wishes to use the signature for the sole purpose of verifying the integrity of the message. In such cases, adding certificate information to the message may be regarded as an unnecessary overhead.

Include Certificate:

This is the default option which places the signatory's certificate inside the XML Signature itself. The following example, shows an example of an XML Signature which has been created using this option:

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
  ...
 <dsig:KeyInfo>
  <dsig:X509Data>
   <dsig:X509SubjectName>CN=Sample User...</dsig:X509SubjectName>
   <dsig:X509Certificate>
       MIIEZDCCA0yg
       ....
       RNp9aKD1fEQgJ
   </dsig:X509Certificate>
  </dsig:X509Data>
 </dsig:KeyInfo>
</dsig:Signature>
```

Expand Public Key:

The details of the signatory's public key are inserted into a `KeyValue` block. The `KeyValue` block is only inserted when this option is checked.

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
  ...
 <dsig:KeyInfo>
  <dsig:X509Data>
   <dsig:X509SubjectName>CN=Sample User...</dsig:X509SubjectName>
   <dsig:X509Certificate>
      MIIE ....... EQgJ
```

```
   </dsig:X509Certificate>
  </dsig:X509Data>
  <dsig:KeyValue>
   <dsig:RSAKeyValue>
    <dsig:Modulus>
       AMfb2tT53GmMiD
       ...
       NmrNht7iy18=
    </dsig:Modulus>
    <dsig:Exponent>AQAB</dsig:Exponent>
   </dsig:RSAKeyValue>
  </dsig:KeyValue>
 </dsig:KeyInfo>
</dsig:Signature>
```

Include Distinguished Name:

If this checkbox is checked, the Distinguished Name of the signatory's X.509 certificate will be inserted in an `<X509SubjectName>` element as shown in the following example:

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
  ...
 <dsig:KeyInfo>
  <dsig:X509Data>
   <dsig:X509SubjectName>CN=Sample,C=IE...</dsig:X509SubjectName>
   <dsig:X509Certificate>
       MIIEZDCCA0yg
       ....
       RNp9aKD1fEQgJ
   </dsig:X509Certificate>
  </dsig:X509Data>
 </dsig:KeyInfo>
</dsig:Signature>
```

Include Key Name:

This option allows you insert a key identifier, or `KeyName`, to allow the recipient to identify the signatory. Enter an appropriate value for the `KeyName` in the **Value** field. Typical values include Distinguished Names (DName) from X.509 certificates, key IDs, or email addresses. Specify whether the specified value is a **Text value** of a **Distinguished name attribute** by checking the appropriate radio button.

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
   ...
 <dsig:KeyInfo>
  <dsig:KeyName>test@ca.com</dsig:KeyName>
 </dsig:KeyInfo>
</dsig:Signature>
```

Put Certificate in an Attachment:

The SOA Security Gateway supports SOAP messages with attachments. By selecting this option, you can save the signatory's certificate to the file specified in the input field. This file can then be sent along with the SOAP message as a SOAP attachment.

From previous examples, it is clear that the user's certificate is usually placed inside a `KeyInfo` element. However, in this example, the certificate is actually contained within an attachment, and not within the XML Signature itself. Clearly, we need a way to reference the certificate from the XML Signature, so that validating applications can process the signature correctly. This is the role of the `SecurityTokenReference` block.

The `SecurityTokenReference` block provides a generic way for applications to retrieve security tokens in cases where these tokens are not contained within the SOAP message. The name of the security token is specified in the `URI` attribute of the `Reference` element.

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
   ...
 <dsig:KeyInfo>
  <wsse:SecurityTokenReference xmlns:wsse="http://schemas.xmlsoap.org/ws/...">
   <wsse:Reference URI="c:\myCertificate.txt"/>
  </wsse:SecurityTokenReference>
 </dsig:KeyInfo>
</dsig:Signature>
```

When the message is actually sent, the certificate attachment will be given a "Content-Id" corresponding to the `URI` attribute of the `Reference` element. The following example shows what the complete multipart MIME SOAP message looks like as it is sent over the wire. It should help illustrate how the `Reference` element actually refers to the "Content-ID" of the attachment:

```
POST /adoWebSvc.asmx HTTP/1.0
Content-Length: 3790
User-Agent: SOA Security Gateway
Accept-Language: en
Content-Type: multipart/related; type="text/xml";
              boundary="----=Multipart-SOAP-boundary"

------=Multipart-SOAP-boundary
Content-Id: soap-envelope
Content-Type: text/xml; charset="utf-8";
SOAPAction=getQuote

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
   ...
 <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
    ...
  <dsig:KeyInfo>
   <ws:SecurityTokenReference xmlns:ws="http://schemas.xmlsoap.org/ws/...">
    <ws:Reference URI="c:\myCertificate.txt"/>
   </ws:SecurityTokenReference>
  </dsig:KeyInfo>
 </dsig:Signature>
    ...
</s:Envelope>

------=Multipart-SOAP-boundary
Content-Id: c:\myCertificate.txt
Content-Type: text/plain; charset="US-ASCII"

MIIEZDCCA0ygAwIBAgIBAzANBgkqhki
....
7uFveG0eL0zBwZ5qwLRNp9aKD1fEQgJ
------=Multipart-SOAP-boundary-
```

Security Token Reference:

A `<wsse:SecurityTokenReference>` element can be used to point to the security token used in the generation of the signature. Select this option if you wish to use this element. The type of the reference must be selected from the **Reference Type** dropdown.

The `<wsse:SecurityTokenReference>`, (within the `<dsig:KeyInfo>`), may contain a `<wsse:Embedded>` security token. Alternatively, the `<wsse:SecurityTokenReference>`, (within the `<dsig:KeyInfo>`), may refer to a certificate via a `<dsig:X509Data>`. Select the appropriate button, **Embed** or **Refer**, depending on whether you want to use an embedded security token or a referred one.

Advanced Tab:

The settings on the **Advanced** tab determine how the recipient's certificate and key are included in the encrypted message.

The recipient's certificate can be embedded into either the encrypted key or a security token reference can be used. Similarly, the recipients distinguished name and/or key name can be included.

The recipients key can be encrypted and included in the encrypted message. The following options are available:

- ***Embed symmetric key inside encrypted data:***

  Place the <xenc:EncryptedKey> inside the <xenc:EncryptedData> element.

- ***Point to symmetric key with Security Token Reference:***

  This option creates a <SecruityTokenReference> in the <EncryptedData> that points to an <EncryptedKey>.

- ***Specify symmetric key via carried keyname:***

  Place the encrypted key's carried keyname inside the <dsig:KeyInfo>/<dsig:KeyName> of the <xenc:EncryptedData>.

- ***Specify symmetric key via retrieval method:***

  Refer to a symmetric key via a retrieval method reference from the <xenc:EncryptedData>.

- ***Symmetric key refers to encrypted data:***

  The symmetric key refers to <xenc:EncryptedData> via a reference list.

- ***Use derived symmetric key:***

  A derived symmetric key is used for the actual encryption. The <enc:EncryptedData> has a <wsse:SecurityTokenReference> to the <wssc:DerivedKeyToken>. The <wssc:DerivedKeyToken> refers to the <enc:EncryptedKey>. Both <wssc:DerivedKeyToken> and <enc:EncryptedKey> are placed inside a <wsse:Security> element.

Advanced Encryption Settings

The **Advanced** on the main **XML-Encryption Settings** screen allows you to configure some of the more complicated settings regarding XML-Encryption. The following settings are available:

Encryption Algorithm:

The following algorithms are available:

- AES-256
- AES-192
- AES-128
- Triple DES

Insert Timestamp:

This option allows you to insert a timestamp as an encryption property.

Generate a Reference List in WS-Security Block:

When this option is selected, a `<xenc:ReferenceList>` that holds a reference to all encrypted data elements is generated. The `<xenc:ReferenceList>` element will be inserted into the WS-Security block indicated by the specified actor.

Use Symmetric Key from Signature:

If this is set to true, it will assume that a previous filter added the `<enc:EncryptedKey>` for the recipient into the message DOM. No `<enc:EncryptedKey>` is added, although the `<enc:EncryptedKey>` from the previous step is edited to include a reference to the new `<enc:EncryptedData>`.

This option is present in order to support WSE 3.0, which stipulates that data must be signed before it is encrypted.

Auto-generation using the XML Encryption Settings Wizard

Since the **XML Encryption Settings** filter must always be used in conjunction with the **XML Encryption** and **Find Certificate** filters, it makes sense to have a wizard that can generate this triplet of filters at the same time. To use this wizard, right-click on the name of the policy in the tree view of the SOA Security Gateway Management Console and select the **XML Encryption Settings** menu option.

For more information on how to configure the **XML Encryption Settings Wizard** please refer to the XML Encryption Settings Wizard help page.

References

XML Encryption Syntax and Processing [http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/]

XML Encryption Requirements [http://www.w3.org/TR/xml-encryption-req]

XML Signature [http://www.w3.org/Signature/]

XML Signature Syntax and Processing [http://www.w3.org/TR/xmldsig-core/]

XPath W3C Recommendation [http://www.w3.org/TR/xpath]

Introduction to Public-Key Cryptography [http://docs.sun.com/source/816-6154-10/]

SOAP Fault

Overview

In cases where a typical SOAP transaction fails, a *SOAP Fault* can be used to convey error information to the SOAP client. The following message shows the format of a SOAP Fault:

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Body>
  <env:Fault>
   <env:Code>
    <env:Value>Receiver</env:Value>
    <env:Subcode>
     <env:Value>policy failed</env:Value>
    </env:Subcode>
   </env:Code>
   <env:Detail xmlns:cafault="http://www.ca.com/soapfaults"
               cafault:type="exception" type="exception"/>
  </env:Fault>
 </env:Body>
</env:Envelope>
```

By default, the SOA Security Gateway returns a very basic SOAP Fault to the client when a message filter has failed. The **SOAP Fault** processor can be added to a policy to return more complicated error information to the client.

For security reasons, it is good practice to return as little information as possible to the client. However, for diagnostic reasons, it is useful to return as much information to the client as possible. Using the **SOAP Fault** processor, administrators have the flexibility to configure just how much information to return to clients, depending on their individual requirements.

SOAP Fault Format

The following configuration options are available in this section:

SOAP Version:

It is possible to send either a SOAP Fault 1.1 or 1.2 response to the client. Select the appropriate version using the radio buttons provided.

Fault Namespace:

Select the default namespace to use in SOAP Faults, or enter a new one if necessary.

Indent SOAP Fault:

If this option is checked, an XSL stylesheet will be run over the SOAP Fault in order to indent nested XML elements. The indented SOAP Fault will be returned to the client.

SOAP Fault Contents

The following configuration options are available in this section:

Show Detailed Explanation of Fault:

If this option is checked, a detailed explanation of the SOAP Fault will be returned within the fault message. This makes it possible to suppress the reason for the exception in a tightly locked down system, i.e. the reason will simply appear as "message blocked" in the SOAP Fault.

Show Filter Exception Path

When this option is checked, the SOA Security Gateway will return a SOAP Fault containing the list of filters run on the message before the error occurred. For each filter listed in the SOAP Fault, the status (i.e. "pass" of "fail") is given. The following message shows a *filter exception path* returned in a SOAP Fault:

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
 </env:Header>
 <env:Body>
  <env:Fault>
   <env:Code>
    <env:Value>Receiver</env:Value>
    <env:Subcode>
     <env:Value>policy failed</env:Value>
    </env:Subcode>
   </env:Code>
   <env:Detail xmlns:cafault="http://www.ca.com/soapfaults"
   cafault:type="exception" type="exception">
    <cafault:path>
     <cafault:visit node="HTTP Parser" status="Pass"></cafault:visit>
     <cafault:visit node="/services" status="Fail"></cafault:visit>
     <cafault:visit node="/status" status="Fail"></cafault:visit>
    </cafault:path>
   </env:Detail>
  </env:Fault>
 </env:Body>
```

```
</env:Envelope>
```

Show Stack Trace

If this option is checked, the SOA Security Gateway will return the Java stack trace for the error to the client. This option should only be enabled under instructions from the CA Support Team.

Show Current Message Attributes

By checking this option, the message attributes present at the time the SOAP Fault was generated will be returned to the client. Each message attribute will form the content of a `<fault:attribute>` element, as illustrated in the following example:

```
<fault:attributes>
  <fault:attribute name="circuit.failure.reason" value="null">
  <fault:attribute name="circuit.lastProcessor" value="HTTP Digest">
  <fault:attribute name="http.request.clientaddr" value="/127.0.0.1:4147">
  <fault:attribute name="http.response.status" value="401">
  <fault:attribute name="http.request.uri" value="/authn">
  <fault:attribute name="http.request.verb" value="POST">
  <fault:attribute name="http.response.info" value="Authentication Required">
  <fault:attribute name="circuit.name" value="Digest AuthN">
</fault:attributes>
```

Customized SOAP Faults

It is possible to create customized SOAP Faults using the **Set Message** filter. The **Set Message** filter can change the contents of the message body to any arbitrary content. When an exception occurs in a policy, it is possible to use this filter to customize the body of the SOAP fault. The following example demonstrates how to generate customized SOAP faults and return them to the client.

Step 1: Create the Top-level Policy:

We will first create a very simple policy called "Main Circuit". This policy will ensure the size of incoming messages is between 100 and 1000 bytes. Messages within this range will be echoed back to the client.

Step 2: Create the Fault Policy

Next, create a second policy called "Fault Circuit". This policy will use the **Set Message** filter to customize the body of the SOAP fault. When configuring this filter, enter the contents of the customized SOAP fault that you want to return to clients in the text area provided.

Step 3: Create a shortcut to the Fault Policy

Add a **Policy Shortcut** filter to the "Main Circuit" and configure it to refer to the "Fault Circuit". Do ***not*** connect this filter to the policy. Instead, right-click on the filter and select the **Set as Fault Handler** menu option. The "Main Circuit" now appears as follows:

So how does it work? Let's assume a 2000-byte message is received by the SOA Security Gateway and is passed to the "Main Circuit" for processing. The message is parsed by the **HTTP Parser** filter and then the size of the message is checked by the **Message Size** filter. Since the message is greater than the size constraints set by this filter, and since there is no failure path configured for this filter, an exception will be thrown. (Please refer to the References section below for more information on policies and how they work.)

When an exception is thrown in a policy, it is handled by the designated *Fault Handler*, if one is present. In the "Main Circuit", we have set a **Policy Shortcut** filter to be the fault handler. This filter delegates to the "Fault Circuit", meaning that when an exception occurs, "Main Circuit" will invoke (or delegate to) the "Fault Circuit".

The "Fault Circuit" consists of 2 filters, which play the following roles:

1. **Set Message:**

   This filter is used to set the body of the message to the contents of the customized SOAP fault.

2. **Reflect**

   Once the SOAP fault has been set to the message body, it will be returned to the client using the **Reflect** filter.

References

SOAP Faults [http://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_Toc478383507]

Logging Configuration

Overview

One of the most important features of a server-based product is its ability to maintain highly detailed and configurable logging. It is crucial that a record of each and every transaction is kept, and that these records can easily be queried by an administrator to carry out detailed transaction analysis. In recognition of this requirement, the SOA Security Gateway provides detailed logging to a number of possible locations.

It is possible to configure the SOA Security Gateway so that it logs information about all requests. Such information includes the request itself, the time of the request, where the request was routed to, and the response that was returned to the client. The actual logging information itself can be written to the console, log file, Unix syslog, and/or a database, depending on what is configured in the **Configure Logging** dialog.

Furthermore, the SOA Security Gateway can digitally sign the logging information it sends to the log files and the database. This means that the logging information cannot be altered after it has been signed, thus enabling an irreversible audit trail to be created. The SOA Security Gateway also ships with a remote logging console called the **Monitoring Console**, which displays real-time logging information.

Configuration

To edit the default logging settings that ship with the SOA Security Gateway, right-click on the Policy Store URL at the root node of the tree in the left hand panel of the SOA Security Gateway Management Console and select the **Default Logging Settings** menu option.

It is possible to override the default log settings at the Process level by right-clicking on the Process in the tree view, selecting **Logging** and then **Custom** from the context menu.

Whether editing the default logging settings or customizing them, the user is presented with a tabbed dialog that makes it possible to configure the SOA Security Gateway to log to the following locations.

Log to File

Select this option to configure the SOA Security Gateway to log to a file. The following

fields should then be configured:

- **File Name:**

  Enter the name of the file that the SOA Security Gateway will log to. By default, the log file is called, "ca". All log files are stored in the `/logs` directory of your SOA Security Gateway installation.

- **File Extension:**

  Enter the file extension of the log file in this field. By default, the log file is given the ".log" extension.

- **File Size:**

  Enter the maximum size that the log file will grow to. When the file reaches the specified limit, a new log file will be created.

- **Log File Signer:**

  In order to sign the log file, select a **User** whose private key will be used in the signing process. By signing the log files, it is possible to verify their integrity at a later stage.

- **Format:**

  It is possible to specify the format of the logging output using the values entered here. Wildcards can be used to output logging information that is specific to the request. The following wildcards can be used:

  - **level:**

    The log level (i.e. fatal, fail, success).

  - **id:**

    The unique transaction ID assigned to the message.

  - **ip:**

    The IP address of the client that sent the request.

  - **t:**

The time that the message was processed in millisecond form.

- **timestamp:**

  The time that the message was processed in user-readable form.

- **filterName:**

  The name of the filter that generated the log message.

- **filterType:**

  The type of the filter that logged the message.

- **text:**

  The text of the log message that was configured in the filter itself. In the case of the **Log Message Payload** filter, the `text` wildcard will contain the message that was sent by the client.

For example, the default logging format is as follows:

```
${level} ${timestamp} ${id} ${text} ${filterType} ${filterName}
```

Log to Database

Using this option, the SOA Security Gateway can be configured to log messages to an Oracle, SQL Server, or MySQL relational database. To configure a new database, click on the **Add** button. To edit or remove an existing database, click on the **Edit** and **Delete** buttons respectively.

Configure the following fields on the **Configure Database Connection** dialog:

- **Name:**

Enter a name for the database.

- **URL:**

  Enter the URL of the database.

- **Username:**

  Enter the username to use to connect to the database.

- **Password:**

  Enter the password for this user.

- **Maximum Connections:**

  The maximum number of connections to keep in a pool of connections to the database.

- **Idle Timeout:**

  Unused idle connections will time out after this number of seconds.

- **Checkout Time:**

  Maximum number of seconds a thread can keep a connection before it is closed and returned to the pool. This is a protection against the thread dying and leaving the connection checked out indefinitely.

- **Maximum Check Out Time:**

  The number of times a connection may be checked out before it is closed. This is used as a safeguard against cursor leak.

- **Use Cache:**

  Specifies whether to cache statements or not.

- **Maximum Retrieval Size:**

  Specifies what is the maximum retrieval size in bytes.

It is possible to sign log messages that are stored in the database to ensure that they can't be tampered with. Select a **User** from the **Database Log Signer** dropdown

whose private key will be used to sign the messages. This user must have been granted the **Sign Log Events** privilege.

Log to Unix Syslog

Check this option to configure the SOA Security Gateway to send logging information to the Unix syslog. Select a previously configured **Syslog Server** from the dropdown. Click the **Add** button to add new syslog server, or **Edit** keycap> to edit an existing server configuration. The following fields should be configured on the **Syslog Server** dialog:

- **Name:**

  Enter a friendly name for the syslog server in the **Name** field.

- **Host:**

  Enter the host name or IP address of the machine where the syslog daemon is running.

- **Facility:**

  Select the syslog facility that the SOA Security Gateway should log to.

Back on the **Syslog** tab of the **Configure Logging** dialog, you must select a server to send log messages to from the **Syslog Server** dropdown. You must also configure the **Format** field as follows:

- **Format:**

  It is possible to specify the format of the log message using the values (including wildcards) entered in this field. Take a look at the section on **Log to File** above for more information on the various wildcards that are available here.

Log to System Console

Check this option to configure the SOA Security Gateway to send logging information to the system console. See the section above for more information on how to configure the

format of the log message using the **Format** field.

Log to Remote Console

Check this option to enable real-time logging to the remote **Monitoring Console**. See the section below for more information on configuring Remote Logging.

Remote Logging

When remote logging has been configured, the SOA Security Gateway will send all log messages to the remote machine, where they will be displayed in the **Monitoring Console**. Remember, remote logging is a privileged operation for which a user must have been given permission. To set up remote logging, follow these instructions:

The Monitoring Console can filter log messages by **Level**, **Message ID**, **Text**, and **Time**. Administrators can use these filters to help identify specific problems in the system.

1. Ensure that **Enable Logging to Remote Console** has been enabled in the **Logging Configuration** dialog.
2. On the remote machine where the **Remote Console** has been installed, start the **Monitoring Console**.
3. By default, the **Monitoring Console** listens for log messages at the following URL:

   http://HOST:8091/admin/adminservlet

   where HOST is the machine on which the Monitoring Console is running.

The p12 file specified here must have been imported into the **Certificate Store** using the **SOA Security Gateway Management Console**, and assigned to a **User**. This user must have been assigned the "Remote Logging" privilege.

The **Monitoring Console** should now be displayed on the remote machine. Try sending a request to the SOA Security Gateway to ensure that logging information is sent to the **Monitoring Console**.

Log Access Filter

Overview

The **Log Access Filter** is used by the SOA Security Gateway to log records of all messages that pass through the filter.

The SOA Security Gateway writes the access log to an `access.log` file in the `log` directory. This file will roll over at the top of the day so that the name of the log file incorporates the date that the log file was created, for example, "access_12Apr2006.log".

The format of the log entries is Common Log Format, which has the following format:

```
host ident authuser date request status bytes
```

The following list explains each item:

- **host:** The remote hostname.
- **ident:** The remote logname of the user.
- **authuser:** The username by which the user has authenticated himself, e.g. the Distinguished Name of a certificate.
- **date:** The data and time of the request.
- **request:** The request line exactly as it originated at the client.
- **status:** The HTTP status code returned to the client.
- **bytes:** The content-length of the document returned to the client.

The following extract from the `access.log` file illustrates the format:

```
m1.ca.com - Good [30/Mar/2008:22:09:05 00] "http://services.com/qotd" 200 587
m3.ca.com - Good [30/Mar/2008:22:10:34 00] "http://services.com/qotd" 200 671
m1.ca.com - Good [30/Mar/2008:22:10:53 00] "http://services.com/qotd" 200 571
.................
.................
```

. . . . . .

Because the **Log Access** filter reports the number of bytes returned to the client (i.e. the "bytes" parameter explained above), it should be positioned towards the end of a policy. A typical policy involving a **Log Access** filter might appear as follows:

Configuration

The **Log Access Filter** requires only a **Name** field to be configured.

Log Message Payload

Overview

The **Log Message Payload** filter is used to log the message payload at any point in the policy. The message payload includes the HTTP headers and MIME/DIME attachments.

By placing the **Log Message Payload** filter at various key locations in the policy, a complete audit trail of the message can be achieved. For example, by placing the filter after each filter in the policy, the complete history of the message can be logged. This is especially useful in cases where the message has been altered by the SOA Security Gateway, for example by signing or encrypting the message, inserting security tokens, or by converting the message to another grammar using XSLT.

Log messages can be stored in several locations, including a database, a file, or the system console.

Configuration

Simply enter a name for the **Log Message Payload** filter in the **Name** field. It is good practice to use descriptive names for these filters. For example, "Log message before signing message", "Log message after signing", would be useful names to give to 2 **Log Message Payload** filters that are placed before and after a **Sign Message** filter.

It is possible to specify the format of the logging output using the values entered in the **Format** field. Wildcards can be used to output logging information that is specific to the request. The following wildcards can be used:

- **level:**

  The log level (i.e. fatal, fail, success).

- **id:**

  The unique transaction ID assigned to the message.

- **ip:**

  The IP address of the client that sent the request.

- **t:**

The time that the message was processed in millisecond form.

- **timestamp:**

  The time that the message was processed in user-readable form.

- **filterName:**

  The name of the filter that generated the log message.

- **filterType:**

  The type of the filter that logged the message.

- **text:**

  The text of the log message that was configured in the filter itself. In the case of the **Log Message Payload** filter, the `text` wildcard will contain the message that was sent by the client.

Log Level and Message

Overview

Each filter in a policy is responsible for logging its own message depending on whether it succeeds, fails, and/or throws an exception. Log messages can be stored in several locations, including a database, a file, or the system console.

A filter will abort when it cannot make the decision it was asked to make. For example, if an LDAP-based authorization filter cannot connect to the LDAP directory it will abort, since it can neither authorize nor refuse access. This is regarded as a *fatal* error.

A filter will fail if returns a false result after carrying out its security processing. Using the authorization example above, the authorization filter will return false if the LDAP directory returns a "not authorized" result to the filter.

Finally, a filter will succeed if it returns a true result after carrying out its processing. For example, if the LDAP directory returns an "authorized" result to the authorization filter, the filter will succeed.

Configuration

The **Log Level and Message** configuration screen is available by clicking the **Next** button on the main screens of all filters.

As stated earlier, it is possible to log messages when the filter succeeds, fails, and/or aborts. Select the **Success**, **Failure**, and/or **Fatal** checkbox(es) to configure the filter to log at the respective level(s).

Default values are provided at each level for all filters. Simply check the checkbox for a particular level to use the default log message for that level. It is possible to specify an alternative log message by entering the message into the text field provided.

All filters *require* and *generate* message attributes, while some *consume* attributes. In some cases it may be useful to log the value of these attributes. For example, instead of an authentication filter logging a generic, "Authentication Failed" message, it is possible to use the value of the `authentication.subject.id` attribute to log the ID of the user that could not be authenticated.

Use the following format to enter a message attribute as a wildcard in a log message. At run-time, the SOA Security Gateway will replace these wildcards with the value of the message attribute

```
${name_of_attribute}
```

For example, to make sure the ID of a non-authenticated user is logged in the message, enter something like the following in the text field for the **Failure** case:

```
The user '${authentication.subject.id}' could not be authenticated.
```

Now if a user with ID "ca" cannot be authenticated by the SOA Security Gateway (i.e. a "fail" case), the following message will be logged:

```
The user 'ca' could not be authenticated.
```

Service Level Agreement (SLA) Filter

Overview

A Service Level Agreement (SLA) is an agreement put in place between a Web Services host and a client of that Web Service in order to guarantee a certain minimum quality of service. It is common to see SLAs in place to ensure that a minimum number of messages result in a communications failure and that responses are received within an acceptable timeframe. In cases where the conditions of the SLA are breached, it is crucial that an alert can be sent to the appropriate party.

The SOA Security Gateway satisfies these requirements by allowing SLAs to be configured at the policy level. It is possible to configure SLAs to monitor the following types of problems:

- Response times
- HTTP status codes returned from the Web Service
- Communication failures

It is important to note that the SLA monitoring performed by the SOA Security Gateway is *statistical*. Because of this, a single message (or even a small number of messages) is not considered a sufficient sample to cause an alert to be triggered. The monitoring engine actually uses an *exponential decay* algorithm to determine whether an SLA is "failing" or not. This algorithm is best explained with an example.

Assume the *poll rate* is set to 3 seconds (i.e. 3000ms), the *data age* is set to 6 seconds (i.e. 6000ms), and you have a Web Service with an average processing time of 100ms. A single client sending a stream of requests through the SOA Security Gateway will be able to generate about 10 requests per second, given the Web Services's 100ms response time.

At every 3 seconds poll period you will have data from a previous 30 samples to consider the average response times of. However, rather than simply using the response time of the *last* 3 seconds worth of data, historical data is "smoothed" into the current estimate of the failing percentage. The new data is combined with the existing data such that it will take approximately the data age time for a sample to disappear from the average.

Therefore the closer the data age is to the sampling rate, the less significant historical data becomes, and the more significant the "last" sample becomes.

In order to generate an alert, you must also have enough significant samples at each

poll period to consider the date to be statistically valid. For example, if a single request arrives over a period of 1 hour it may not be fair to say that "less than 20%" of all received requests have failed the response time requirements. For this reason, statistical analysis provides a more realistic SLA monitoring mechanism than a solution based purely on absolute metrics.

Response Time Requirements

It is possible to monitor the response times of Web Services protected with the **SLA Filter**. The filter offers 8 ways in which to measure response times:

Response Time Measurement	***Meaning***
receive-request-start	The time that the SOA Security Gateway receives the first byte of the request from the client.
receive-request-end	The time that the SOA Security Gateway receives the last byte of the request from the client.
send-request-start	The time that the SOA Security Gateway sends the first byte of the request to the Web Service.
send-request-end	The time that the SOA Security Gateway sends the last byte of the request to the Web Service
receive-response-start	The time that the SOA Security Gateway receives the first byte of the response from the Web Service
receive-response-end	The time that the SOA Security Gateway receives the last byte of the response from the Web Service.
send-response-start	The time that the SOA Security Gateway sends the first byte of the response to the client.
send-response-end	The time that the SOA Security Gateway sends the last byte of the response to the client.

The SOA Security Gateway will measure each of the 8 time values. They will available

for processing after the policy has completed for a single request. These 8 options are available for the following reasons:

- The SOA Security Gateway may start to send the first byte to the Web Service before the last byte is received from the client, i.e. send-request-start <receive-request-end. This will occur if the invoked policy does not require the full message to be read into memory.
- The SOA Security Gateway may start to send the response to the client before the complete response has been received from the Web Service, i.e. send-response-start < receive-response-end. This will occur when invoked policy does not require the full message to be read into memory.
- It is possible that the Web Service may start to send the response before it has received the complete request. However, the SOA Security Gateway will not start to read the response until it has sent the complete request. This means that the following is always true:- send-request-end < receive-response-start.
- The time value for send-response-end will depend upon the client application. This value will be larger if the client is slow to read the response.

To add a Response Time Requirement for an SLA, click the **Add** button.

To configure the start time and end time for the response time measurement, click the **Add** button. On the **Settings** tab, specify the percentage of response times that must be below a specified time interval (in milliseconds) in the fields provided. The purpose of these options is to allow for situations where a very small number of unusually slow requests may cause an SLA to trigger unnecessarily. By using percentages, such requests will not distort the statistics collected by the SOA Security Gateway.

Click on the **Message Text** tab to configure the messages that will appear in the alert message when the SLA is breached and also when the SLA is cleared, i.e. when the breached conditions are no longer in breach of the SLA.

Finally, click on the **Advanced** tab to configure timing information. Select a **Start Timing Point** from the 8 times listed in the table above. The SOA Security Gateway will *start* measuring the response time from this time. Then select an **End Timing Point** from the 8 times listed in the table above. The SOA Security Gateway will *stop* measuring the response time from this time.

HTTP Status Requirements

HTTP status codes may be received from a Web Service. The SOA Security Gateway can

be configured to monitor these and generate alerts based on the number of occurrences of certain types of status code response. HTTP status codes are 3 digit codes that may be grouped into standard status "classes", with the first digit indicating the status class. The status classes are as follows:

Class of HTTP Status Code	***Meaning***
1xx	These status codes indicate a provisional response.
2xx	These status codes indicate that the client's request was successfully received, understood, and accepted.
3xx	These status codes indicate that further action needs to be taken by the user agent in order to fulfill the request.
4xx	These status codes are intended for cases in which the client seems to have erred. For example 401, means that authentication has failed.
5xx	These status codes are intended for cases where the server has encountered an unexpected condition that prevented it from fulfilling the request. For example, 500 is used to transmit SOAP faults.

The SOA Security Gateway may monitor a class (i.e. range) of status codes, or they may monitor specific status codes. For example, it is possible to configure the following HTTP status code requirements:

- At least 97% of the requests must yield HTTP status codes between 200 and 299
- At most 2% of requests may yield HTTP status codes between 400 and 499
- At most 0% of requests may yield HTTP status code 500

Click the **Add** button in the **HTTP Status Code Requirements** section.

Select an existing status code or class of status codes from the **HTTP Status Code** dropdown. To add a new code or range of codes, click the **Add** button.

Enter a name for the new code or range of codes in the **Name** field of the **Configure HTTP Status Code** dialog. Enter the *first* HTTP status code in the range of status codes

that you want to monitor in the **Start Status** field. Then enter the *last* HTTP status code in the range of status codes that you want to monitor in the **End Status** field.

If you just want to monitor one specific status code, enter the same code in the **Start Status** and **End Status** fields.

Click **OK** when you are satisfied with the selected range of status codes to return to the previous dialog. The remaining 2 fields allow the administrator to specify the minimum or maximum percentage of received HTTP status codes that fall into the configured range before an alert is triggered.

Again, the use of percentages here is to allow for situations where a very small number of requests return the status codes within the "forbidden" range. By using percentages, such requests will not distort the statistics collected by the SOA Security Gateway.

Click on the **Message Text** tab to configure the messages that will appear in the alert message when the SLA is breached and also when the SLA is cleared, i.e. when the breached conditions are no longer in breach of the SLA.

Communications Failure Requirements

The SOA Security Gateway is deemed to have experienced a *communications failure* when it fails to connect to the Web Service, fails to send the request, or fails to receive the response.

The requirements for communications failures may be expressed as follows:

- No more than 4% of requests may result in communications failures.

Enter the percentage of allowable communications failures in the field provided. An alert will be configured if the percentage of communicates failures rises above this level.

Click on the **Message Text** tab to configure the messages that will appear in the alert message when the SLA is breached and also when the SLA is cleared, i.e. when the breached conditions are no longer in breach of the SLA.

Select Alerting System

If an alert is triggered, it must be sent to an alerting destination. The SOA Security Gateway can send alerts to the following destinations:

- Windows Event Log
- Email Recipient
- SNMP Network Management System

- Unix Syslog
- CheckPoint's FireWall-1

The **Alert System** table at the bottom of the **SLA Filter** screen displays all available alerting destinations that have been configured through the alerting interface .

The administrator should select one or more alerting systems. An alert will be sent to each selected system in the event of a violation of the performance requirements. Alert clearances will be generated when the violation no longer exists.

Routing Configuration

Overview

The purpose of this guide is to describe how to configure the SOA Security Gateway to send messages to external Web Services. The SOA Security Gateway offers a number of different filters that can be used to route messages. Depending on how the SOA Security Gateway is perceived by the client, different combinations of routing filters can be used.

For example, the SOA Security Gateway can act both as a proxy and an endpoint (i.e. in-line) server for a client, depending on how the client is configured. In each case the request received by the SOA Security Gateway appears slightly different, a fact that the SOA Security Gateway can take advantage of when routing the message onwards. Furthermore, the SOA Security Gateway can provide *service virtualization* by shielding the underlying hierarchy of back-end Web Services from clients.

In the next section, we will explain how clients can use the SOA Security Gateway as both a proxy and as an endpoint server. We will then go on to show how *service virtualization* works. Once these basic concepts have been explained, we will help you to identify the combination of routing filters that is best suited to your deployment scenario.

Proxy or Endpoint Server

The SOA Security Gateway can be used by clients as both a proxy sever and an endpoint server. When a client uses the SOA Security Gateway as a proxy server, it sends up the complete URL of the destination Web Service in the HTTP request line. The SOA Security Gateway can use the URL to determine the host and port to route the message to. The following HTTP request shows an example of a request received by the SOA Security Gateway when acting as a proxy for a client:

```
POST http://localhost:8080/services/getHello HTTP/1.1
```

Alternatively, when the SOA Security Gateway is acting as an endpoint (i.e. in-line) server, the client sends the request directly to the SOA Security Gateway. In this case, the request line appears as follows:

```
POST /services/getHello HTTP/1.1
```

In this case, only the path on the server is specified and no scheme, host, or port number is included in the HTTP request line. Since this information is not provided by the client, the SOA Security Gateway must be explicitly configured to route the message on to the specific destination.

Service Virtualization

It is sometimes desirable to shield the underlying structure of the directory hierarchy in which Web Services reside from external clients. This can be done by providing a mapping between the path that the client accesses and the actual path at which the Web Service resides.

For example, suppose we have 2 Web Services accessible at the URIs, `/helloService/getHello` and `/financeService/getQuote`. We may want to hide the fact that these services are actually deployed under different paths, perhaps exposing them under a common `/services` base URI, for example, `/services/getHello` and `/services/getQuote`. The client is therefore unaware of the underlying hierarchy (e.g. directory structure) of the 2 Web Services. This is called *service virtualization*.

Choosing the Correct Routing Filters

The purpose of this section is to identify, first, how your clients perceive the SOA Security Gateway, and second, to determine whether or not you wish to virtualize your back-end Web Services. Depending on these requirements, different combinations of routing filters can be used, as outlined in the 4 "cases" described in the subsequent sections.

In order to determine the combination of routing filters that is most appropriate for your scenario, it is necessary to consider your answers to the following simple questions:

Proxy or Endpoint

- Is the client using the SOA Security Gateway as a proxy? If so, then you will be interested in cases 1 or 2 below, depending on whether *service virtualization* is required or not.
- Alternatively, is the client using the SOA Security Gateway as the endpoint of the connection, i.e. as an in-line server? If so, then you should look at cases 3 or 4 below.

Service Virtualization

- Do you want to shield the actual hierarchy of protected Web Services by exposing a *virtual* view of these services? If so, then cases 2, 4, and 5 below will be of interest.
- Is *service virtualization* important? If not, then you should refer to cases 1 and 3 below.

The above permutations can be summarized in the following table:

Proxy or Endpoint	***Service Virtualization***	***Example***
Proxy	No	Case 1: Proxy without Virtualization
Proxy	Yes	Case 2: Proxy with Virtualization
Endpoint	No	Case 3: Endpoint without Virtualization
Endpoint	Yes	Case 4: Endpoint with Virtualization
Proxy or Endpoint	Yes	Case 5: Simple Redirect

Case 1: Proxy without Service Virtualization

In this case, the SOA Security Gateway is configured as a HTTP proxy for the client and will maintain the original path used by the client in the HTTP request. So for example, if the SOA Security Gateway is listening at `http://localhost:8080/`, and the Web Service is running at `http://localhost:5050/services/getQuote`, then the request line of the client HTTP request will appear as follows:

```
POST http://localhost:5050/services/getQuote HTTP/1.1
```

Since the client has been configured to use the SOA Security Gateway instance running on localhost at port 8080 as its HTTP proxy, the client will automatically send all messages to the proxy. However, it will include the full URL of the ultimate destination of

the message in the request line of the HTTP request.

When the SOA Security Gateway receives the request, it extracts this URL from the request line and uses it to determine the destination of the message. In the above example, the SOA Security Gateway will route the message onwards to `http://localhost:5050/services/getQuote`.

The following policy can be configured to route the message to the URL specified in the request line of the client request:

The following table explains the role of each filter in the policy. For more information on a particular filter, click on the appropriate link in the **Help** column.

Filter	***Role in Policy***	***Help***
Dynamic Router	Extracts the URL of the destination Web Service from the request line of the incoming HTTP request. The **Dynamic Router** is usually used when the SOA Security Gateway is perceived as a proxy by the client.	Dynamic Router
Connection	Responsible for establishing the connection to the destination Web Service and sending the message over this connection. This connection can be mutually authenticated if necessary.	Connection

Case 2: Proxy with Service Virtualization

In the second case, the SOA Security Gateway is also perceived as a HTTP proxy by the client. However, this time the SOA Security Gateway exposes a *virtualized* view of the services that it protects, i.e. *service virtualization*.

To achieve this, the SOA Security Gateway must provide a mapping between the path used by the client and the actual path under which the service is deployed. Assuming the SOA Security Gateway is running at `http://localhost:8080/services` and the Web Service is deployed at `http://localhost:5050/financialServices/quotes/getQuote`, the following is an example of what the client may send up in the HTTP request line:

```
POST http://localhost:5050/services/getQuote HTTP/1.1
```

To achieve this, the SOA Security Gateway must provide a mapping between what the client requests, i.e. `/services/getQuote`, and the actual address of the Web Service, i.e. `/financialServices/quotes/getQuote`. The **Rewrite URL** filter in the following policy fulfills this role:

The following table explains the roles of the 3 filters within the policy:

Filter	***Role in Policy***	***Help***
Dynamic Router	Extracts the URL of the destination Web Service from the request line of the incoming HTTP request. The **Dynamic Router** is usually used when the SOA Security Gateway is perceived as a proxy by the client.	Dynamic Router
Rewrite URL	Specifies the mapping between the path requested by the client and the actual path under which the Web Service is deployed, therefore providing service virtualization.	Rewrite URL
Connection	Responsible for establishing the connection to the destination Web Service and sending the message over this connection. This connection can be mutually authenticated if necessary.	Connection

Case 3: Endpoint without Service Virtualization

In this scenario, the client sees the SOA Security Gateway as the endpoint to its con-

nection, and the SOA Security Gateway must be configured to route messages on to a specific destination. So, for example, assuming that the SOA Security Gateway is running at `http://localhost:8080/services`, the request line of the client's HTTP request will be received by the SOA Security Gateway as follows:

```
POST /services HTTP/1.1
```

It is clear from the request line above that no information about the scheme, host, or port of the destination Web Service is specified. Therefore, this information must be configured in the SOA Security Gateway so that it knows where to route the message on to. The **Static Router** allows the user to enter connection details for the destination Web Service.

Assuming that the Web Service is running at `http://localhost:5050/stockquote/getPrice`, the host, port, and scheme respectively are as follows: "localhost", "5050", and "http". This information must be explicitly configured in the **Static Router**. The following policy illustrates this scenario:

The following table explains the role of each filter in the above policy:

Filter	***Role in Policy***	***Help***
Static Router	Allows the user to explicitly specify the host, port, and scheme at which the Web Service is listening. This filter must be used when the client sees the SOA Security Gateway as the endpoint to its connection, i.e. the SOA Security Gateway is not acting as a proxy for the client.	Static Router
Connection	Responsible for establishing the connection to the destination Web Service and sending the message over this connection. This connection can be mutually authenticated if necessary.	Connection

Case 4: Endpoint with Service Virtualization

In the last case, the SOA Security Gateway is acting as the endpoint to the client connection (i.e. it is not acting as a proxy) and is hiding the deployment hierarchy of protected Web Services from clients. In other words, it is performing *service virtualization*.

In this scenario, the client will send messages directly to the SOA Security Gateway. So, for example, assuming that the SOA Security Gateway is running at `http://localhost:8080/services`, and the Web Service is running at `http://localhost:5050/stockquote/getPrice`, the request line of the client's HTTP request will be received by the SOA Security Gateway as follows:

```
POST /services HTTP/1.1
```

The **Static Router** must then be configured to route the message on to port 8080 on localhost using the http scheme, while the **Rewrite URL** filter provides the mapping between the path requested by the client (i.e. `/services` and the path under which the Web Service is deployed (i.e. `/stockquote/getPrice`). The following policy illustrates a sample policy that will provide *service virtualization* when the SOA Security Gateway is used as an endpoint:

The following table explains the role of each filter in the policy:

Filter	***Role in Policy***	***Help***
Static Router	Allows the user to explicitly specify the host, port, and scheme at which the Web Service is listening. This filter must be used when the client sees the SOA Security Gateway as the endpoint to its connection, i.e. the SOA Security Gateway is not acting as a proxy for the client.	Static Router
Rewrite URL	Provides the mapping between the path requested by the client and the path under which the Web Service is deployed.	Rewrite URL

Filter	*Role in Policy*	*Help*
Connection	Responsible for establishing the connection to the destination Web Service and sending the message over this connection. This connection can be mutually authenticated if necessary.	Connection

Case 5: Simple Redirect

In some cases, it is necessary for the SOA Security Gateway to simply route the incoming message to an entirely different URL. The **Rewrite URL** filter can be used for this purpose, in addition to simple rewriting the path on which the request was received as described in cases 2 and 4 above. Note that the full URL of the destination Web Service should be specified in this case in the **Rewrite URL** filter.

The following policy illustrates the use of the **Redirect URL** filter to route messages to a fully qualified URL:

The following table describes the role of each filter in the policy above:

Filter	*Role in Policy*	*Help*
Rewrite URL	Used to specify the fully qualified URL of the destination Web Service.	Rewrite URL
Dynamic Router	In this case, the **Dynamic Router** is used to parse the URL specified in the **Rewrite URL** filter into its constituent parts. The HTTP scheme, port, and host of the Web Service are extracted and set to the internal message object for use by the **Connection** filter.	Dynamic Router
Connection	Responsible for establishing the connection to the destination Web Service and sending the message over this connection. This con-	Connection

Filter	*Role in Policy*	*Help*
	nection can be mutually authenticated if necessary.	

Summary

The following points illustrate the key concepts to consider when configuring the SOA Security Gateway to connect to external Web Services:

- The **Connection** filter **must** always be used since it is responsible for establishing the connection to the Web Service.
- *Service virtualization* can be achieved using the **Rewrite URL** filter.
- If the client is configured to use the SOA Security Gateway as a proxy, the SOA Security Gateway can use the **Dynamic Router** to extract the URL from the request line of the HTTP request. It can then route the message on to this URL.
- If the client sees the SOA Security Gateway as the endpoint of the connection (i.e. not as a proxy), then the **Static Router** must be used to explicitly configure the host, port, and scheme of the destination Web Service.
- Take a look at the Connection Index for more information on the different connection filters provided by the SOA Security Gateway.

Dynamic Router

Overview

The SOA Security Gateway can act as a proxy for clients of the secured Web Service. When a client uses a proxy, it includes the fully qualified URL of the destination in the request line of the HTTP request. It sends this request to the configured proxy, who then forwards the request to the host specified in the URL. The relative path used in the original request will be preserved by the proxy on the outbound connection.

The following is an example of a HTTP request line that was made through a proxy, where WEB_SERVICE_HOST is the name or IP address of the machine hosting the destination Web Service:

```
POST http://WEB_SERVICE_HOST:80/myService HTTP/1.0
```

When the SOA Security Gateway is acting as a proxy for clients, it can receive requests like the one above. The Dynamic Router filter can simply route the request onwards to the URL specified in the request line, i.e. http://WEB_SERVICE_HOST:80/myService.

Configuration

Enter a name for the router in the **Name** field on the **Dynamic Router** filter configuration screen.

Connection

Overview

The **Connection** is responsible for making the connection to the remote Web Service. It relies on connection details that are set by the other filters in the **Routing** filter group.

Since the **Connection** filter actually connects out to other services, it is responsible for negotiating the SSL handshake involved in setting up a (mutually) authenticated secure channel.

General Configuration

Enter a name for the filter in the **Name** field. Click on the tabs to configure the various aspects of the **Connection** filter.

Trusted Certificates

This section lists all CA certificates that have been imported into the CA **Certificate Store**. Select the certificates that the SOA Security Gateway will consider to be trusted when it attempts to establish a connection to a remote server. The remote server's SSL certificate must be issued by one of the certificates selected (i.e. trusted) on this tab.

Client SSL Authentication

The SOA Security Gateway can be configured to authenticate itself to the remote Web Service. It will do so using the certificate chosen on this tab.

HTTP Authentication

The SOA Security Gateway can use both HTTP basic and HTTP digest authentication to authenticate to the remote server. In both cases the **User Name** and **Password** of a user must be specified in the fields provided.

Advanced

This tab allows you to configure certain advanced features of the **Connection** filter.

The following configuration options are available:

Send via Proxy

Check this option if the SOA Security Gateway must connect to the destination Web

Service through a HTTP proxy. In this case, the SOA Security Gateway will include the full URL of the destination Web Service in the request line of the HTTP request. For example, if the destination Web Service resides at `http://localhost:8080/services`, then the request line would appear as follows:

```
POST http://localhost:8080/services HTTP/1.1
```

If the SOA Security Gateway was not routing through a proxy, the request line would appear as follows:

```
POST /services HTTP/1.1
```

Ciphers:

The **Ciphers** field allows the administrator to specify the ciphers that the server supports. The server will send this list of supported ciphers to the destination server, which will then select the highest strength common cipher as part of the SSL handshake. The selected cipher will then be used to encrypt the data as it is sent over the secure channel.

HTTP Host Header

A HTTP 1.1 client ***must*** send a `Host` header in all HTTP 1.1 requests. The `Host` header identifies the hostname and port number of the requested resource as specified in the original URL given by the client.

When routing messages on to target Web Services, the SOA Security Gateway can either forward on the `Host` as received from the client, or it can specify the address and port number of the destination Web Service in the `Host` header that it routes onwards.

Select **Use Host header specified by client** to force the SOA Security Gateway to always forward on the original `Host` header that it received from the client. Alternatively, to configure the SOA Security Gateway to include the address and port number of the destination Web Service in the `Host` header, select the **Generate new Host header** radio button.

HTTP Status Code

Overview

This filter is responsible for setting the HTTP status code on response messages. This allows the administrator to ensure that a more meaningful response is sent to the client in the case of an error or anomaly occurring in a configured policy.

So, for example, if a **Relative Path** filter failed, it may be useful to return a "503 Service Unavailable" response. Similarly, if a user does not present identity credentials when attempting to access a protected resource, the SOA Security Gateway could be configured to return a "401 Unauthorized" response to the client.

HTTP status code are returned in the *status-line* of a HTTP response. The following examples are typical examples:

```
HTTP/1.1 200 OK
HTTP/1.1 400 Bad Request
HTTP/1.1 500 Internal Server Error
```

Configuration

Enter a name for the filter in the **Name** field on the **HTTP Status** filter configuration screen.

Enter the status code that will be returned to the client in the **HTTP response code status** field. Please refer to the HTTP Specification [http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html] for a complete list of status codes.

Messaging System

Overview

A *messaging system* is a loosely coupled, peer-to-peer facility where clients can send messages to, and receive messages from any other client. A messaging system differs from an email system in that the communication takes place between software components, as opposed to people, or people and software components.

In a messaging system, a client sends a message to a messaging agent. The recipient of the message can then connect to the same agent and read the message from it. However, the sender and recipient of the message do not have to be available at the same time in order to communicate (unlike HTTP, for example). The sender and recipient need only know the name and address of the messaging agent to talk to. In this way, messaging systems are said to be loosely coupled.

The Java Messaging System (JMS) is an implementation of such a messaging system. It provides an API for creating, sending, receiving, and reading messages. Java-based applications can use it to connect to other messaging system implementations.

A *JMS provider* can deliver messages synchronously and asynchronously, meaning that the client can either "fire and forget" messages or wait for a response before resuming processing. Furthermore, the JMS API can ensure different levels of reliability in terms of message delivery. For example, it can ensure that the message is delivered once and only once, or at least once.

The SOA Security Gateway uses the JMS API to connect to other messaging systems that expose a JMS interface, including IBM's MQSeries, JBoss Messaging, BEA's WebLogic Server, TIBCO's EMS, IBM's WebSphere Server, and SonicMQ from Sonic Software.

Note: It is important to note that the JMS provider's jar files must be added to the SOA Security Gateway's classpath in order for this filter to function properly. If the provider's implementation is platform-specific, copy the provider's jar files to the `INSTALL_DIR/ext/PLATFORM` folder, where "INSTALL_DIR" points to the root of your product installation and "PLATFORM" represents the platform on which the SOA Security Gateway has been installed, i.e. "win32", "Linux.i386", or "SunOS.sun4u-32". If the provider's implementation is platform-independent, the jar files can be placed in the `INSTALL_DIR/ext/lib` folder.

Request

The **Request** tab is used to configure properties of the request to the messaging system. The following fields can be configured:

JMS Session:

Select an existing JMS session from the dropdown list. JMS sessions are configured globally at the Process level. Take a look at the Messaging System help page for more information on how to configure JMS sessions so that they can be used in this Connection filter.

Destination:

Enter the name of the JMS queue or topic that you want to drop messages on to.

Delivery Mode:

The SOA Security Gateway supports 2 delivery modes: persistent and non-persistent.

1. **Persistent:**

   Instructs the JMS provider to ensure that a message is not lost in transit if the JMS provider fails. A message sent with this delivery mode is logged to persistent storage when it is sent.

2. **Non-persistent:**

   Does not require the JMS provider to store the message. With this mode, the message may be lost if the JMS provider fails.

Priority Level:

You can use message priority levels to instruct the JMS provider to deliver urgent messages first. The ten levels of priority range from 0 (lowest) to 9 (highest). If you do not specify a priority level, the default level is 4. A JMS provider tries to deliver higher priority messages before lower priority ones but does not have to deliver messages in exact order of priority.

Time to Live:

By default, a message never expires. However, if a message will become obsolete after a certain period you may want to set an expiration time (in milliseconds). If the specified time to live value is 0, the message never expires.

Message ID:

Enter an identifier to be used as the unique identifier for the message. By default, the unique identifier is the ID assigned to the message by the SOA Security Gateway, i.e. `${id}`. However, it is possible to use a proprietary correlation system, perhaps using MIME message IDs instead of CA message IDs.

Correlation ID:

Enter an identifier for the message that the SOA Security Gateway will use to correlate response messages with the corresponding request messages. Usually, if `$id` is specified in the **Message ID** field above, it will also be used here to correlate request messages with their correct response messages.

Auto MIME:

If this option is checked, messages will be formatted according to the SOAP over JMS [http://mail-archives.apache.org/mod_mbox/ws-axis-dev/200701.mbox/raw/%3C80A43FC052CE3949A327527DCD5D6B27020FB65C@MAIL01.bedford.progress.com%3E/3] proposal. This option is checked by default.

Custom Message Properties:

It is possible to set custom properties for messages in addition to those provided by the header fields. Custom properties may be required to provide compatibility with other messaging systems. It is possible to use message attributes as property values. For example, it is possible to create a property called "AuthNUser" and set its value to ${authenticated.subject.id}. Other applications can then filter on this property, e.g. only consume messages where "AuthNUser" equals "admin".

Response

The **Response** tab is used to configure whether the SOA Security Gateway should use asynchronous or synchronous communication when talking to the messaging system. If the SOA Security Gateway is to use asynchronous communication then click on the "Fire and forget" radio button. If synchronous communication is required, select the "Wait for response" option.

When synchronous communication is selected, the SOA Security Gateway will wait on a message from a queue/topic from the messaging system. The SOA Security Gateway will set the "JMSReplyTo" property on each message that it sends. The value of the "JMSReplyTo" property will be the queue, temporary queue, topic, or temporary topic that was selected in the **Response Type** dropdown. It is up to the application that consumes the message from the queue to send the message back to the destination specified in "JMSReplyTo".

The SOA Security Gateway will set the "JMSCorrelationID" property to the value of the **Correlation ID** field on the **Response** tab to correlate requests messages to their corresponding response messages. If the user has selected to use a temporary queue or temporary topic then this will be created when the SOA Security Gateway is started up.

Response Type:

Select where the response message is to be placed. If the response is to be read from an existing queue or topic then select "Place response in queue or Topic". If a temporary destination is to be used for reading the response then select "Response in temporary queue". Alternatively, if no response is expected, select the "No response" option.

When a temporary destination is selected, this destination will be created at start-up of the SOA Security Gateway. Only the SOA Security Gateway will be able to read from the temporary destination however any application can write to it. The SOA Security Gateway will use the value of the "JMSReplyTo" header to indicate the location where it will read responses from.

Destination:

If a queue or topic is to be used, its name should be entered here.

Time Out:

The SOA Security Gateway will wait a certain time period for a response to be received before it will time out. If the SOA Security Gateway does time out waiting for a response, this filter will fail. Enter the time out value in milliseconds.

References

Java Messaging Service [http://java.sun.com/products/jms/tutorial/1_3_1-fcs/doc/jms_tutorialTOC.html]

SOAP over JMS Proposal [http://mail-archives.apache.org/mod_mbox/ws-axis-dev/200701.mbox/raw/%3C80A43FC052CE3949A327527DCD5D6B27020FB65C@MAIL01.bedford.progress.com%3E/3]

Rewrite URL

Overview

The **Rewrite URL** filter is used to specify the path on the remote machine to send the request to. It is usually used in conjunction with a **Static Router**, whose role is to supply the host and port of the remote service.

Configuration

Configure the following fields on the **Rewrite URL** filter configuration screen.:

Name:

Enter a name for the filter in the **Name** field.

URL:

Enter the relative path of the Web Service in the **URL** field. The SOA Security Gateway will combine the path specified here with the host and port number specified in the **Static Router** filter in order to build up the complete URL to route to.

Alternatively, it is possible to perform simple URL rewrites by specifying a fully qualified URL into the **URL** field. A **Dynamic Router** can then be used to route the message to the specified URL.

Static Router

Overview

The SOA Security Gateway uses the information configured in the **Static Router** to connect to a machine that is hosting a Web Service. The **Static Router** should be used in conjunction with a **Rewrite URL** filter in order to specify the path to send the message to on the remote machine.

Configuration

The following fields must be configured on the **Static Router** configuration screen.:

Name:

Enter a name for the filter.

Host:

Enter the host name or IP address of the remote machine that is hosting the destination Web Service.

Port:

Enter the port on which the remote service is listening.

HTTP:

Select this option if the SOA Security Gateway should send the message to the remote machine over plain HTTP.

HTTPS:

Select this option if the SOA Security Gateway should send the message to the remote machine over a secure channel using SSL. A **Connection** filter can be used to configure the SOA Security Gateway to mutually authenticate to the remote system.

Read WS-Addressing

Overview

The WS-Addressing specification defines a transport-independent standard for including addressing information within SOAP messages. The SOA Security Gateway can read WS-Addressing information contained within a SOAP message and subsequently use this information to route the message to its intended destination.

Configuration

Complete the following fields to configure the SOA Security Gateway to read WS-Addressing information contained within a SOAP message.

Name:

Enter a name for the filter here.

Address Location:

Specify the name of the element within the WS-Addressing block that contains the address of the destination server to which the SOA Security Gateway will route the message.

By default, XPath expressions are available to extract the destination server from the "From", "To", "ReplyTo", and "FaultTo" elements. Click the **Add** button to add a new XPath expression to extract the address from a different location.

Insert WS-Addressing

Overview

The WS-Addressing specification defines a transport-independent standard for including addressing information within SOAP messages. The SOA Security Gateway can generate WS-Addressing information based on a configured endpoint within a policy, and then insert this information into SOAP messages.

Configuration

Complete the following fields to configure the SOA Security Gateway to insert WS-Addressing information into a SOAP message.

Name:

Enter a name for the filter here.

To:

The message will be delivered to the destination entered here.

From:

Informs the destination server where the message originated from.

Reply To:

Indicates to the destination server where it should send response messages to.

Fault To:

Indicates to the destination server where it should send fault messages to.

MessageID:

A unique identifier to distinguish this message from others at the destination server. It also provides a mechanism for correlating a specific request with its corresponding response message.

Action:

The action entered here indicates what action the destination server should take on the message.

Relates To:

If responses are to be received asynchronously, the value entered here provides a method to associate an incoming reply to its corresponding request.

Namespace:

The WS-Addressing namespace to use in the WS-Addressing block.

Call Internal Service

Overview

The **Call Internal Service** filter is a special filter that passes messages to an internal Servlet Application that has been deployed at the SOA Security Gateway. The appropriate application is selected based on the relative path on which the request message was received.

The filter is used by "Management Services" that are configured to listen on the Management Interface on port 8090. For more information on how the **Call Internal Service** filter is used by these services, please refer to the Note on Management Services in the Configuring HTTP Services help page.

Wait for Response Packets

Overview

Packet Sniffers are a type of Passive Service. Rather than opening up a TCP port and *actively* listening for requests, the Packet Sniffer *passively* reads data packets off the network interface. The Sniffer assembles these Packets into complete messages that can then be passed into an associated policy.

Because the Packet Sniffer operates passively (i.e. does not listen on a TCP port) and, therefore, completely transparently to the client, it is most useful for monitoring and managing Web Services. For example, the Sniffer can be deployed on a machine running a Web Server acting as a container for Web Services. Assuming that the Web Server is listening on TCP port 80 for traffic, the Packet Sniffer can be configured to read all packets destined for port 80 (or any other port, if necessary). The packets can then be marshalled into complete HTTP/SOAP messages by the Sniffer and passed into a policy that logs the message to a database, for example.

Packet Sniffer Configuration

Since Packet Sniffers are mainly for use as passive monitoring agents, they are usually created within their own Service Group. So, for example, a new group can be created for this purpose by right-clicking on the Process in the tree view of the SOA Security Gateway Management Console, selecting the **Add Service Group** menu option, and the entering "Packet Sniffer Group" on the **Add Service Group** dialog.

We can then add a Relative Path Service to this Group by right-clicking on the "Packet Sniffer Group" and selecting the **Add Relative Path** menu option. Enter a path in the field provided and select the policy that you want to dispatch messages to when the Packet Sniffer detects a request for this path (after it assembles the packets). So, for example, if the Relative Path is configured as "/a", and the Packet Sniffer assembles packets into a request for this path, the request will be dispatched to the policy selected in the Relative Path Service.

Finally, we can add the Packet Sniffer itself by right-clicking on the "Packet Sniffer Group" node, selecting **Packet Sniffer**, and then the **Add** menu option. Complete the following fields on the **Packet Sniffer** dialog:

Device to Monitor:

Enter the name or identifier of the network interface that the Packet Sniffer will monitor. The default entry here is "any", but it is important to note that this is only valid on Linux. On UNIX-based systems, network interfaces are usually identified using names like "eth0", "eth1", and so on. On Windows, these names are more complicated, for ex-

ample, "\Device\NPF_{00B756E0-518A-4144 ... }.

Filter:

The Packet Sniffer can be configured to only intercept certain types of packets. For example, it can ignore all UDP packets, only intercept packets destined for port 80 on the network interface, ignore packets from a certain IP address, listen for all packets on the network, and so on.

The Packet Sniffer uses the **libpcap** library filter language to achieve this. This language has a complicated but powerful syntax that allows you to *filter* what packets are intercepted and what packets are ignored. As a general rule, the syntax consists of one or more expressions combined with conjunctions, such as "and", "or", and "not". The following table lists a few examples of common filters and explains what they actually filter:

Filter Expression	*What does it filter?*
port 80	Capture only traffic for the HTTP Port (i.e. 80).
host 192.168.0.1	Capture traffic to and from IP address 192.168.0.1.
tcp	Capture only TCP traffic.
host 192.168.0.1 and port 80	Capture traffic to and from port 80 on IP address 192.168.0.1.
tcp portrange 8080-8090	Capture all TCP traffic destined for ports from 8080 through to 8090.
tcp port 8080 and not src host 192.168.0.1	Capture all TCP traffic destined for port 8080 but not from IP address 192.168.0.1.

The default filter of "tcp" simply captures all TCP packets arriving on the network interface. For more information on how to configure filter expressions like these, please refer to the man pages of **tcpdump** man page, which can be found here [http://www.tcpdump.org/tcpdump_man.html].

Promiscuous Mode:

When listening in "promiscuous mode", the Packet Sniffer will capture all packets on the same Ethernet network, regardless of whether or not the packets are addressed to the network interface that the Sniffer is monitoring.

Sniffing Response Packets

The SOA Security Gateway can capture both incoming and outgoing packets when it is listening passively (i.e. not opening any ports) on the network interface. Say, for example, a Web Service is deployed in a web server that listens on port 80. The SOA Security Gateway can be installed on the same machine as the web server. It is configured *not* to open any ports and to use a Packet Sniffer to capture all packets destined for TCP port 80.

When packets arrive on the network interface that are destined for this port, they are assembled by the Packet Sniffer into HTTP messages and passed into the configured policy. Typically this policy logs the message to an audit trail, and so usually consists of just a **Log Message** filter.

Assuming that we also want to log response messages passively, as is typically required for a complete audit trail, we can use the **Wait for Response Packets** filter to correlate response packets with their corresponding requests. The **Wait for Response Packets** filter assembles the response messages into HTTP messages and can then log them again using the **Log Message Payload** filter. The following circuit will log both request and response messages captured transparently by the Packet Sniffer:

You can see from the circuit that the first logging filter logs the *request* message. By this stage, the Packet Sniffer has assembled the request packets into a complete HTTP request, and this is what is passed to the "Log Request Message" filter.

The "Assemble response packets" filter is a **Wait for Response Packets** filter that assembles response packets into complete HTTP response messages and passes them to the "Log Response Message" filter, which logs the complete response message. More information on the **Log Message Payload** filter is available in the Log Message Payload help page.

Abort Filter

Overview

The **Abort** filter can be used to force a policy to throw an exception. It can be used to test the behavior of the policy when an exception occurs.

For example, to quickly test how the policy behaves when a **Message Size** filter throws an exception, it is possible to place an **Abort** filter before it in the policy. The following policy diagram illustrates the setup:

Configuration

Enter a name for the filter in the **Name:** field.

Copy/Modify Attributes

Overview

The **Copy/Modify Attributes** filter copies the values of message or user attributes to other message or user attributes. It is also possible to set the value of a message or user attribute to a user-specified value.

Configuration

The configured attribute-copying rules are listed in the table. To add a new rule, click the **Add** button.

The **Copy/Modify Attributes** screen can be used to copy a message or user attribute to a different message or user attribute. The **From Attribute** represents the source attribute, while the **To Attribute** represents the destination attribute.

The attribute value can be copied from 3 possible sources:

- **Message:**

  Select this option to copy the value of a message attribute. The name of the source attribute should be specified in the **Name:** field.

- **User:**

  This option should be selected if a user attribute stored in the `attribute.lookup.list` is to be copied. Enter the name (and namespace if the attribute was extracted from a SAML attribute assertion) of the user attribute in the **Name** and **Namespace** fields.

  If there are multiple values stored in the `attribute.lookup.list` for the attribute entered in the **Name** field, only the first value will be copied.

- **Value:**

  Select this option to copy a user-specified value to an attribute. Enter the new attribute value in the **attribute** field. It is possible to enter a wildcard here to generically represent the value of a message attribute, as opposed to entering a specific value directly. The syntax for entering wildcards is as follows:

```
${authentication.subject.id}
```

In this case the value of the `authentication.subject.id` attribute is copied to the named attribute.

The message can be copied to either of 2 types of attributes:

- **Message:**

  The attribute can be copied to any message attribute. The name of the attribute should be specified in the **Name** field.

- **User:**

  Select this option if the attribute or value should be copied to a user attribute stored in the `attribute.lookup.list`. Specify the name and namespace (if necessary) of this attribute in the **Name** and **Namespace** fields.

  If there are multiple values stored in the `attribute.lookup.list` for the attribute entered in the **Name** field of the **From attribute** section, the attribute value will be copied to the first occurrence of the attribute name in list.

Check the **Create list attribute** checkbox if the new attribute can contain several items, i.e. it is a list.

False Filter

Overview

The **False** filter can be used to force a path in the policy to return false. This can be useful in cases where you want to create a *false positive* path in a policy.

The following policy parses the HTTP request and then runs a **Message Size** filter on the message to make sure that the message is no larger than 1000 bytes. If we want to make sure that the message cannot be greater than this size, we can connect a **False** filter to the *success* path of the **Message Size** filter. This means that an exception will be raised if a message exceeds 1000 bytes in size.

Configuration

Enter a name for the filter in the **Name:** field.

Pause Filter

Overview

The **Pause** filter is mainly used for testing purposes. A **Pause** filter will cause a policy to sleep for a specified amount of time.

Configuration

Enter a name for the filter in the **Name:** field. The policy will sleep for the time specified in the **Pause for** field. The sleep time is specified in milliseconds.

Reflect Filter

Overview

The **Reflect** filter echoes the HTTP request headers, body, and attachments back to the client.

Configuration

Enter a name for the filter in the **Name** field. Specify a HTTP status response code to return to the client in the **HTTP Response Code Status** field.

Reflect Message And Attributes Filter

Overview

The **Reflect Message and Attributes** filter echoes the HTTP request headers, body, and attachments back to the client. It also echoes back the message attributes that were stored in the message at the time when the message completed the policy.

Configuration

Enter a name for the filter in the **Name:** field.

Scripting Language Filter

Overview

This filter allows you to write bespoke JavaScript code to interact with the message as it is processed by the SOA Security Gateway. It is possible to get, set, and evaluate specific message attributes with this filter.

This user must implement the `invoke` method. This method takes a CA `Message` object as a parameter and returns a boolean result. Some typical uses of the **Scripting Filter** are as follows:

- Check the value of a specific message attribute
- Set the value of a message attribute
- Remove a message attribute
- DOM processing on the XML request or response

The SOA Security Gateway ships with a **Script Library** that contains a number of pre-written invoke methods to manipulate specific message attributes. For example, there are invoke methods to check the value of the SOAPAction header, to remove a specific message attribute, and another to assign a particular role to a user.

Any Java code can be used within the chunk of script provided the required classes/packages are imported and placed on the SOA Security Gateway's classpath. This can be done by copying the jar file containing the Java classes to the `INSTALL_DIR/lib` directory of your SOA Security Gateway installation.

For a complete list of available message attributes, please refer to the Message Attribute Reference page, which can be found in the References section of the documentation.

The reader is also advised to take a look at the script examples that ship with the product. These can be found in the **Script Library**, which can be accessed by clicking the **Show script library** button on the filter's main configuration screen.

Configuration

The JavaScript code should be written in the **Script** text area. A JavaScript function skeleton is displayed by default. You should use this skeleton code as the basis for your JavaScript code.

It is also possible to load an existing script from the **Script Library** by clicking on the **Load script library** button.

Click on any of the **Configured Scripts** in the table to display the script in the text area provided. A script can be edited directly in the **Script** text area. Make sure to click the **Update** button to store the updated script to the **Script Library**.

New scripts can be added by clicking on the **Add** button, which displays the **Script Details** dialog.

Enter a **Name** and a **Description** for the new script in the fields provided. The script can then be written in the **Script** text area.

When writing the JavaScript code, it is important to note the following:

- The `invoke` method **must** be implemented.
- The `invoke` method takes a CA `Message` object as a parameter, and returns a boolean.
- It is possible to obtain the value of a message attribute using the `getProperty` method of the `Message` object. Take a look at the Message Attribute Reference for a complete list of attributes.

Trace Filter

Overview

The **Trace** filter traces the current message attributes to the configured trace destination(s).

Configuration

Enter a name for the filter in the **Name:** field.

True Filter

Overview

The **True** filter can be used to force a path in the policy to return true. This can be useful in cases where you want to prevent a path from ending on a false case and consequently throwing an exception.

The following policy parses the HTTP request and then runs "Attachment1" on the message. If "Attachment1" passes, the message will be echoed back to the client by the "Reflect" filter. If it fails, however, the "Attachment2" filter will be run on the message. Since this is an *end* node, if this filter fails, an exception will be thrown.

By adding a **True** filter to the "Attachment2" filter, this path will always end on a "true" case, and so will no throw an exception if "Attachment2" fails.

Configuration

Enter a name for the filter in the **Name:** field.

HTTP Parser

Overview

The **HTTP Parser** parses the HTTP request headers and body. As such, it acts as a "barrier" in the policy to guarantee that the entire content has been received before any other filters are invoked. It requires the `content.body` attribute.

The **HTTP Parser** filter forces the server to do "store-and-forward" routing instead of the default "cut-through" routing, where the request is only parsed on-demand. This filter can be used as a simple test to ensure that the message is XML, for example.

Configuration

Enter a name for the filter in the **Name** field.

Quote of the Day

Overview

The **Quote of the Day** filter is a useful test utility for returning a simple SOAP response to a client. The SOA Security Gateway will wrap the quote in a SOAP response, which can then be returned to the client.

Configuration

Simply enter the quote in the **Quotes** textarea. This quote can be returned in a SOAP response to the client by setting the **Reflect** filter to be the successor of this filter in the policy.

The **Quote of the Day** filter can also load a file containing a list of quotes at run-time. In this case, a random quote from the file will be returned to the client in the SOAP response. Each quote should be delimited by a '%' character on a newline. This is analogous to the *BSD fortune format*. The format of this file is shown in the following example:

```
Most powerful is he who has himself in his own power.
%
All science is either physics or stamp collecting.
%
A cynic is a man who knows the price of everything and the value of nothing.
%
Intellectuals solve problems; geniuses prevent them.
%
If you can't explain it simply, you don't understand it well enough.
```

The quotes can, of course, be simply entered in this format into the **Quotes** textarea to achieve the same goal.

The following example shows a SOAP response returned by the SOA Security Gateway to a client who requested the **Quote of the Day** service:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
                   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body xmlns:ca="www.ca.com">
```

```
    <ca:getQuoteResponse>
      Every cloud has a silver lining
    </ca:getQuoteResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Set Response Status

Overview

The **Set Response Status** filter is used to explicitly set the response status of a call. This status is then recorded as a message metric for use in reporting.

This filter is primarily used in cases where the Fault handler for a policy is actually a Policy Shortcut. If the Policy Shortcut passes, the overall fail status still exists. The **Set Response Status** filter can then be used to explicitly set the response status back to "pass", if necessary.

This filter should only be used under very rare circumstances and under advice from the CA support team.

Configuration

Enter a name for the filter in the **Name:** field. Then select the appropriate response status using the **Response Status** radio button.

Configuration Web Service

Overview

This filter is only used by the configuration Management Service and should not need to be configured.

Set Web Service Context

Overview

The **Set Web Service Context** filter is used in a policy to determine the service from the Web Service Repository to take resources from. For example, by pointing this filter at a pre-configured "getQuote" service in the Web Service Repository, the policy will then know to return the WSDL for this particular service when a WSDL request is received. The **Return WSDL** filter is used in conjunction with this filter in order to achieve this. Take a look at the Web Services Repository for a detailed example.

This filter is configured automatically when auto-generating a policy from a WSDL file and is not normally manually configured. Take a look at the Web Services Repository for more information on how to auto-generate a policy from a WSDL file.

Configuration

Name:

Enter a name for the filter in the **Name:** field.

Web Service:

Select a service from the Web Service Repository by checking the box next to the desired service. For more information on adding services to the Web Services Repository please refer to the Web Services Repository help page.

Return WSDL

Overview

The **Return WSDL** filter returns a WSDL file from the Web Services Repository. This filter is configured automatically when auto-generating a policy from a WSDL file and is not normally manually configured. Take a look at the Web Services Repository for more information on how to auto-generate a policy from a WSDL file.

Configuration

Enter a name for the filter in the **Name:** field.

Certificate Chain Check

Overview

Whenever the SOA Security Gateway receives a client's X.509 certificate, either in an XML Signature or as part of an SSL handshake, it needs to determine whether or not that certificate can be trusted. For example, it is a trivial task for a user to generate a structurally sound X.509 certificate. This certificate can then be used to negotiate mutually authenticated connections to publicly available services.

Clearly, this scenario represents a security nightmare for IT administrators - we can't just allow any user to generate their own certificate and use it on the Internet. The server must be able to *trust* the authenticity of the client certificate. Furthermore, it must be able to verify that the certificate originated from a trusted source. To do this a server can perform a *certificate chain check* on the client certificate.

The main purpose of certificate chain validation is to ensure that a certificate has been issued by a trusted source. Typically, in a Public-Key Infrastructure (PKI), a Certificate Authority (CA) is responsible for issuing and distributing certificates. The whole infrastructure is based on the premise of *transitive trust* - if everybody trusts the CA, then everybody transitively trusts the certificates issued by that CA. If entities only trust certificates that have been issued by the CA, they can then reject certificates which have been self-generated by clients.

When a CA issues a certificate, it digitally signs the certificate and inserts a copy of its own certificate into it. This is called a certificate chain. Whenever an application (such as the SOA Security Gateway) receives a client certificate it can extract the issuing CA's certificate from it, and run a certificate chain check to determine whether or not it should trust the CA. If it trusts the CA, it will also trust the client certificate.

The question then begs itself - how does the SOA Security Gateway *trust* a Certificate Authority? The SOA Security Gateway maintains a repository of both trusted CA certificates, and trusted server certificates for use in SSL communications. In order to trust a certain CA, that CA's certificate must be imported into the **CA Trusted Certificate Store**.

Configuration

The **SOA Security Gateway Management Console** provides an easy-to-use interface for configuring certificate chain validation. This interface allows you to amalgamate CA and server certificates into groups such that if an incoming client certificate has been issued by any of the CAs in the group, the SOA Security Gateway will trust the certificate. Simply enter a name for the group in the **Group Name** field. To populate the new group, simply click the **Add/Edit** button.

By selecting a group from this dropdown, the members of this group will be displayed in the **Certificate Alias** table. To add and/or remove members from the selected group, click the **Add/Edit** button.

Certificates can be added to and removed from new or existing groups using the **Configure Trusted Certificate Groups** dialog which is displayed on clicking the **Add/Edit** button.

The **Configure Trusted Certificate Groups** dialog consists of 2 main tables. The first table lists all certificates currently in the **Trusted Certificate Store**, i.e. those that are trusted by the SOA Security Gateway. The second table lists the members of the group selected in the **Group Name** field.

To add a certificate to a trusted group, simply select it from the **Certificate Store** table, and click the **Add ->** button. The certificate will now appear in the group certificates table. Similarly to remove a certificate from the group, select it from the group certificates table and click the **<- Remove** button. The certificate will now be removed from the group table.

It is also possible to add, remove, and view certificates in the **Trusted Certificate Store** using this dialog. To add a certificate to the **Trusted Certificate Store**, click the **Add** button, which displays the **Import Certificate** dialog.

Browse to the location of the CA certificate file, and enter an **Alias** for the certificate. This **Alias** will be used to uniquely identify the certificate within the SOA Security Gateway.

A certificate can be removed by simply selecting the certificate in the **Trusted Store** table, and then clicking the **Remove** button. The certificate will be removed from the table, and will no longer be trusted by the SOA Security Gateway.

Finally, it is also possible to examine the details of any one of the certificates in the **Trusted Certificate Store**. To do this, again select a certificate from the **Trusted Certificate** table, and then click on the **View** button.

Certificate Validation

Overview

Whenever the SOA Security Gateway receives an X.509 certificate, either as part of the SSL handshake or as part of the XML message itself, it is important to be able to determine whether that certificate is legitimate or not. Certificates can be revoked by their issuers if it becomes apparent that the certificate is being used maliciously. Such certificates should never be trusted, and so it is very important that the SOA Security Gateway can perform certificate validation.

The SOA Security Gateway uses the following methods/protocols to validate certificates:

OCSP - Online Certificate Status Protocol

OCSP is an automated certificate checking network protocol. The SOA Security Gateway can query the OCSP responder for the status of a certificate. The responder returns whether the certificate is still trusted by the CA that issued it.

CRL - Certificate Revocation Lists

A CRL is a signed list indicating a set of certificates that are no longer considered valid (i.e. revoked certificates) by the certificate issuer. The SOA Security Gateway can query a CRL to find out if a given certificate has been revoked - if the certificate is present in the CRL, it should not be trusted.

XKMS - XML Key Management Services

XKMS is an XML-based protocol for (amongst other things) establishing the trustworthiness of a certificate over the Internet. The SOA Security Gateway can query an XKMS responder to determine whether or not a given certificate can be trusted or not.

Configuration

The SOA Security Gateway can check that the validity of a client certificate using any of the following methods:

1. OCSP - Online Certificate Status Protocol
2. CRL - Certificate Revocation Lists
3. XKMS - XML Key Management Services

Note:- In order to validate a certificate using either an or CRL lookup, the issuing CA's certificate should be trusted by the SOA Security Gateway. This is because for a CRL lookup, the CA's public key is needed to verify the signature on the CRL, and for an OCSP request, the protocol stipulates that the CA's public key must be submitted as part of the request. The issuing CA's public key is not always present in issued certificates, so it is necessary to retrieve it from the SOA Security Gateway's certificate store instead.

OCSP - Online Certificate Status Protocol

1. Enter or select a name for the validation rule in the **Name** field.
2. Select *OCSP* from the **Type** dropdown.
3. Optionally enter a description of the rule in the **Description** field.
4. Select a group of OCSP Responders from the **URL Group** field. The SOA Security Gateway will attempt to connect to the Responders in the selected group in a round-robin fashion. It will attempt to connect to the Responders with the highest priority first, before connecting to Responders with a lower priority. URL Groups can be added, edited, and removed by selecting the **Add**, **Edit**, and **Remove** buttons respectively.

   Take a look at the Configuring URL Groups section below for more information on adding and editing URL groups.
5. Enter the user name of a User whose key will be used to sign status requests sent to the OCSP responder in the **User Name** field. This user must have been assigned the **Sign OCSP or XKMS requests** privilege. This can be done through the **Users** interface.
6. Enter the corresponding password for this user in the **Password** field.
7. If the OCSP Responder signs the OCSP response, and you wish to validate this signature, select the **Validate Response** checkbox.

CRL - Certificate Revocation Lists

1. Enter or select a name for the validation rule in the **Name** field.
2. Select *CRL* from the **Type** dropdown.
3. Optionally enter a description of the rule in the **Description** field.
4. Select a previously configured LDAP directory from the **LDAP directory** dropdown list, or add a new one using the **Add** button.

XKMS - XML Key Management Services

1. Enter or select a name for the validation rule in the **Name** field.
2. Select *XKMS* from the **Type** dropdown.
3. Optionally enter a description of the rule in the **Description** field.
4. Enter the URL of the XKMS Responder in the **URL** field.
5. Enter the user name of a User whose key will be used to sign status requests sent to the XKMS responder in the **User Name** field. This user must have been assigned the **Sign OCSP or XKMS requests** privilege. This can be done through the **Users** interface.
6. Enter the corresponding password for this user in the **Password** field.

Configuring URL Groups

The SOA Security Gateway can make connections on a round-robin basis to the URLs listed in a URL group, thus enabling a high degree of failover to external servers. URL groups can be configured by selecting the **Add** and/or **Edit** buttons.

The SOA Security Gateway will attempt to connect to the listed servers according to the priorities assigned to them. So, for example, let's assume there are two "High" priority URLs, one "Medium" URL, and a single "Low" URL configured. Assuming the SOA Security Gateway can successfully connect to the two "High" priority URLs, it will alternate requests between these two URLs only in a round-robin fashion. The other group URLs will not be used at all. If, however, both of the "High" priority URLs become unavailable, the SOA Security Gateway will then try to use the "Medium" priority URL, and only if this fails will the "Low" priority URL be used.

So, in general, the SOA Security Gateway will attempt to round-robin requests over URLs of the same priority, but will use higher priority URLs before lower priority ones. When a new URL is added to the group it is automatically given the highest priority. Priorities can then be changed by selecting the URL and clicking the **Up** and **Down** buttons.

Individual URLs can be added and edited by selecting the URL from the table and clicking on the **Add** and **Edit** buttons respectively.

The following fields should be completed:

- **URL:**

Enter the full URL of the external server.

- **Timeout:**

  Specify the timeout in seconds for connections to the specified server.

- **Time:**

  Whenever the server becomes unavailable for whatever reason (maintenance, for example), no attempt will be made to connect to that server until the time specified here has elapsed. In other words, once a connection failure has been detected, the next connection to that URL will be made after this amount of time.

- **Username:**

  If the specified server requires clients to authenticate to it over 2-way SSL, a **User** must be selected here for authentication. This user must have been assigned the "Use for client authentication" privilege.

- **Password:**

  Enter the password for this user.

- **Host/IP:**

  If the specified server sits behind a proxy server, the host name or IP address of the proxy server must be entered here.

- **Port:**

  Enter the port on which the proxy is listening.

Database Connection

Overview

The details entered on this dialog tell the SOA Security Gateway how to connect to the database. The SOA Security Gateway maintains a JDBC pool of database connections to avoid the overhead of setting up and tearing down connections to service simultaneous requests. This pool is implemented by the *Jakarta DBCP (Database Connection Pools)*. The settings in the **Advanced** section of this screen configure the connection pool. Take a look at the DBCP Configuration table below to see how the fields correspond to specific configuration DBCP settings.

Configuration

The following fields should be completed on this screen:

Name:

Enter a name for the database connection in the **Name** field.

URL:

Enter the fully qualified URL of the location of the database.

User Name:

The username to use to access the database.

Password:

The password for the user specified in the **User Name** field.

Initial Size:

The value entered here determines the initial size of the DBCP pool when it is first created.

Maximum Number of Active Connections:

The maximum number of active connections that can be allocated from the JDBC pool at the same time. The default maximum is 8 active connections.

Maximum Number of Idle Connections:

The maximum number of active connections that can remain idle in the pool without extra ones being released. The default maximum is 8 connections.

Minimum Number of Idle Connections:

The minimum number of active connections that can remain idle in the pool without extra ones being created. The default minimum is 8 connections.

Maximum Wait Time:

The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. or -1 to wait indefinitely. The default time is 10000ms and a value of -1 indicates an indefinite time to wait.

Time Between Eviction:

The number of milliseconds to sleep between runs of the thread that evicts unused connections from the JDBC pool.

Number of Tests:

The number of connection objects to examine from the pool during each run of the evictor thread. The default number of objects is 3.

Minimum Idle Time:

The minimum amount of time an object may sit idle in the pool before it is eligible for eviction by the idle object evictor (if any).

DBCP Configuration

The table below shows the correspondence between the fields on the **Database Configuration** screen and the DBCP configuration properties:

Field Name	***DBCP Configuration Property***
URL	url
User Name	username
Password	password
Initial Size	initialSize
Maximum Number of Active Connections	maxActive
Maximum Number of Idle Connections	maxIdle

Field Name	***DBCP Configuration Property***
Minimum Number of Idle Connections	minIdle
Maximum Wait Time	maxWait
Time Between Eviction	timeBetweenEvictionRunsMillis
Number of Tests	numTestsPerEvictionRun
Minimum Idle Time	minEvictableIdleTimeMillis

References

Jakarta DBCP Configuration [http://jakarta.apache.org/commons/dbcp/configuration.html]

Database Query

Overview

The **Database Query** allows you to enter a SQL query, stored procedure, or function call that the SOA Security Gateway will run in order to return a specific user's profile from a database.

Configuration

The following fields should be completed on this screen:

Name:

Enter a name for this database query here.

Database Query:

Enter the actual SQL query, stored procedure, or function call (i.e. "the query") in the textarea provided. When executed, the query should return a single user's profile. The following are examples of SQL statements and stored procedures:

```
select * from users where username=${authentication.subject.id}
{ call load_user (${authentication.subject.id}, ${out.param}) }
{ call ${out.param.cursor} := p_test.f_load_user(${authentication.subject.id}) }
```

It is clear from the examples above that wildcards in the form of message attributes can be used in the query. The wildcard that is most commonly used in this context is the `authentication.subject.id`, which holds the identity of the authenticated user. Take a look at the Message Attribute Reference for a complete list of message attributes.

Statement Type:

The database can take the form of a SQL query, stored procedure, or function call as demonstrated in the above examples. Select the appropriate radio button depending on whether the database query is a SQL **Query** or a **Stored procedure/function call**

Table Structure:

In order to process the resultset that is returned by the database query, the SOA Security Gateway needs to know whether the user's attributes are structured as rows or columns in the database table.

The following example of a database table shows the user's attributes (i.e. "Role", "Dept", and "Email") structured as table columns:

Username	***Role***	***Dept***	***Email***
Admin	Administrator	Engineering	admin@org.com
Tester	Testing	QA	tester@org.com
Dev	Developer	Engineering	dev@org.com

In the following table, the user's attributes have been structured as name-value pairs in table rows:

Username	***Attribute Name***	***Attribute Value***
Admin	Role	Administrator
Admin	Dept	Engineering
Admin	Email	admin@org.com
Tester	Role	Testing
Tester	Dept	QA
Tester	Email	tester@org.com
Dev	Role	Developer
Dev	Dept	Engineering
Dev	Email	dev@org.com

If the user's attributes are structured as column names in the database table, then select the **attributes as column names** radio button. If, on the other hand, the attributes are structured as name-value pair in table rows, then select the **attribute name-value pairs in rows** option.

Configuring LDAP Directories

General Configuration

A filter that uses an LDAP directory to authenticate a user or retrieve attributes for a user must have an LDAP directory associated with it. The **Configure LDAP Server** screen is used to configure connection details of the LDAP directory.

When a filter that uses an LDAP directory is run for the first time after a server refresh/restart, the server will bind to the LDAP directory using the connection details configured on the **Configure LDAP Server** dialog. Usually the connection details include the username and password of an administrator user who has read access to all users in the LDAP directory for whom we wish to retrieve attributes or authenticate.

To configure LDAP connection information:

1. Select or enter a name for the LDAP filter in the **Filter Name** dropdown list.
2. Enter the location of the LDAP directory in the **URL** field. The URL is a combination of the protocol (LDAP), the IP address of the host machine and the port number for the LDAP service. By default, port 389 is reserved for LDAP connections. The following is an example of a valid LDAP directory URL:

   ```
   ldap://192.168.0.45:389
   ```

Authentication Configuration

If the configured LDAP directory requires clients to authenticate to it, you must select the appropriate authentication method in the **Authentication Type** field. When the SOA Security Gateway connects to the LDAP directory, it will be authenticated using the selected method. The SOA Security Gateway can authenticate to an LDAP directory using the following methods:

- None
- Simple
- Digest-MD5

It is important to note that if any of the following methods are to connect to the LDAP server over SSL, then that server's SSL certificate must be imported into the **CA Trus-**

ted Certificate Store.

None:

No authentication credentials need be submitted to the LDAP server for this method. In other words, the client connects anonymously to the server. Typically a client is only allowed to perform "read" operations when connected anonymously to the LDAP server. It is not necessary to enter any details for this authentication method.

Simple:

Simple authentication involves sending a user name and corresponding password in clear-text to the LDAP server. Since the password is passed in clear-text to the LDAP server, it is recommended to connect to the server over an encrypted channel, for example, over SSL.

It is not necessary to specify a **Realm** for the *Simple* authentication method. The realm is only used when a hash of the password is supplied (i.e. for Digest-MD5). However, in cases where the LDAP server contains multiple realms, and the specified user name is present in more than one of these realms, then it is at the discretion of the specific LDAP server as to which user name will actually *bind* to it.

Click the **SSL Enabled** checkbox to force the SOA Security Gateway to connect to the LDAP directory over SSL. In order to successfully establish SSL connections with the LDAP directory, the directory's certificate must be imported into the SOA Security Gateway's certificate store.

Digest-MD5:

With *Digest-MD5* authentication, the server generates some data and sends it to the client. The client encrypts this data with its password according to the MD5 algorithm. The LDAP server then uses the client's stored password to decrypt the data and hence authenticate the user.

The **Realm** field is optional here, but may be necessary in cases where the LDAP server contains multiple realms. If a realm is specified here, the LDAP server will attempt to authenticate the user for the specified realm only.

Signature Location

Overview

It is possible for a given XML message to contain several XML Signatures. For example, consider an XML document (e.g. a company policy approval form) which must be digitally signed by a number of users (e.g. department managers) before being submitted to the ultimate Web Service (e.g. a company policy approval Web Service). Such a message will contain several XML Signatures by the time it is ready to be submitted to the Web Service.

In such cases, where multiple signatures will be present within a given XML message, it is necessary to specify which signature the SOA Security Gateway should use in the validation process.

Configuration

The SOA Security Gateway can extract the signature from an XML message using several different methods. The signature can be extracted:

- Using WS-Security Actors
- From the SOAP header
- Using XPath

Select the most appropriate method from the **Signature Location** dropdown. Your selection will depend on the types of SOAP messages that you expect to receive. For example, if incoming SOAP messages will contain an XML Signature within a WS-Security block, you should choose this option from the dropdown.

Using WS-Security Actors:

If the signature is present in a WS-Security block:

1. Select *WS-Security block* from the **Signature Location** dropdown list.
2. Select a SOAP Actor from the **Select Actor/Role(s)** dropdown. Each Actor uniquely identifies a separate WS-Security block. By selecting *Current actor only* from the dropdown, the WS-Security block with no Actor will be taken.
3. In cases where there may be multiple signatures within the WS-Security block, it is

necessary to extract one using the **Signature Position** field.

The following is a skeleton version of a message where the XML Signature is contained within the *sample* WS-Security block, i.e. `soap-env:actor="sample"`.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
<soap-env:Header>
 <wsse:Security xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"
                    soap-env:actor="sample">
  <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
    ....
  </dsig:Signature>
 </wsse:Security>
</soap-env:Header>
<soap-env:Body>
    <ns1:getTime xmlns:ns1="urn:timeservice">
        </ns1:getTime>
  </soap-env:Body>
</soap-env:Envelope>
```

SOAP Header:

If the signature is present in the SOAP Header:

1. Select *SOAP message header* from the **Signature Location** dropdown list.
2. If there is more than one signature in the SOAP Header, then it is necessary to specify which signature the SOA Security Gateway should use. Specify the appropriate signature by setting the **Signature Position** field.

The following is an example of an XML message where the XML Signature is contained within the SOAP header:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
  <soap-env:Header>
    <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
      ....
    </dsig:Signature>
  </soap-env:Header>
  <soap-env:Body>
    <ns1:getTime xmlns:ns1="urn:timeservice">
        </ns1:getTime>
```

```
  </soap-env:Body>
</soap-env:Envelope>
```

Using XPath:

Finally, an XPath expression can be used to locate the signature.

1. Select *Advanced (XPath)* from the **Signature Location** dropdown list.
2. Select an existing XPath expression from the dropdown, or add a new one by clicking on the **Add** button. XPath expressions can also be edited or removed with the **Edit** and **Remove** buttons respectively.

The default *First Signature* XPath expression takes the first signature from the SOAP Header. The expression is as follows:

XPath Expression:	
/ / soap:Envelope/soap:Header/dsig:Signature[1]	

To edit this expression, click the **Edit** button to display the **Enter XPath Expression** dialog.

An example of a SOAP message containing an XML Signature in the SOAP header is provided below. The following XPath expression instructs the SOA Security Gateway to extract the first signature from the SOAP header:

XPath Expression:	
/ / soap:Envelope/soap:Header/dsig:Signature[1]	

Since the elements referenced in the expression (i.e. `Envelope` and `Signature`) are *prefixed* elements, it is necessary to define the namespace mappings for each of these elements as follows:

Prefix	***URI***
soap	http://schemas.xmlsoap.org/soap/envelope/
dsig	http://www.w3.org/2000/09/xmldsig#

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
  <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
     ....
  </dsig:Signature>
  </soap:Header>
  <soap:Body>
    <product xmlns="http://www.ca.com">
      <name>SOA Product*</name>
      <company>Company</company>
      <description>Web Services Security</description>
    </product>
  </soap:Body>
</soap:Envelope>
```

When adding your own XPath expressions, you must be careful to define any namespace mappings in a manner similar to that outlined above. This avoids any potential clashes that might occur where elements of the same name, but belonging to different namespaces are present in an XML message.

What Must Be Signed

Overview

The **What Must Be Signed** section allows the administrator to define the exact content that must be signed in order for a SOAP message to pass the corresponding filter. The purpose of this filter is to ensure that the client has signed something meaningful (i.e. part of the SOAP message) as opposed to some arbitrary data that would pass a "blind" signature validation.

This prevents clients from simply pasting technically sound, but unrelated signatures into messages in the hope that they will pass any blind signature verification. For example, the user may be able to generate a valid XML Signature over any arbitrary XML document. Then, by including the signature and XML portion into a "malicious" SOAP message, the signature will pass a blind signature validation, and the harmful XML will be allowed to reach the Web Service.

The **What Must Be Signed** section ensures that clients must sign a part of the SOAP message, and therefore prevents them from pasting arbitrary XML Signatures into the message.

Configuration

An XPath expression is used to identify the nodeset (i.e. the series of elements) that must be signed. To specify that nodeset, select either an existing XPath expression from the **XPath Expression** dropdown list, or add a new one using the **Add** button. XPath expressions can also be edited and removed with the **Edit** and **Remove** buttons respectively.

Enter or select an XPath expression in the **What must be signed** dialog.

An example of a SOAP message is provided below. The following XPath expression indicates that all the contents of the SOAP body, including the `Body` element itself, should be signed:

```
/soap:Envelope/soap:Body/descendant-or-self::node()
```

You would also have to supply the namespace mapping for the `soap` prefix, for example:

Prefix	***URI***
soap	ht- tp://schemas.xmlsoap.org/soap/envelope/

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <product xmlns="http://www.ca.com">
      <name>SOA Product</name>
      <company>Company</company>
      <description>Web Services Security</description>
    </product>
  </soap:Body>
</soap:Envelope>
```

Configuring XPath Expressions

Overview

The SOA Security Gateway uses XPath expressions in a number of ways, for example, to locate an XML Signature in a SOAP message, to determine what elements of an XML message to validate against an XML Schema, to check the content of a particular element within an XML message, amongst many more uses.

There are two ways to configure XPath expressions on this screen:

1. Manual Configuration
2. XPath Wizard

1. Manual Configuration

If you are already familiar with XPath and wish to configure the expression manually, complete the following fields, using the examples below if necessary:

1. Enter or select a name for the XPath expression in the **Name** dropdown.
2. Enter the XPath expression to use in the **XPath Expression** field.
3. In order to resolve any prefixes within the XPath expression, the namespace mappings (i.e. **Prefix**, **URI**) should be entered in the table.

Let's take a look at an example. Consider the following SOAP message:

```
 <?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
  <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="sample">
      ..............
      ..............
      ..............
      ..............
  </dsig:Signature>
</soap:Header>
  <soap:Body>
    <prod:product xmlns:prod="http://www.ca.com">
      <prod:name>SOA Product*</prod:name>
      <prod:company>Company</prod:company>
      <prod:description>WebServices Security</prod:description>
    </prod:product>
```

```
  </soap:Body>
</soap:Envelope>
```

The following XPath expression evaluates to true if the <*name*> element contains the value *SOA Security Gateway*:

XPath Expression: //prod:name[text()='SOA Security Gateway']

In this case it is necessary to define a mapping for the *prod* namespace as follows:

Prefix	***URI***
prod	http://www.ca.com

Let's look at another example. This time the element that is to be examined belongs to a default namespace. Consider the following SOAP message:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
  <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="sample">
      ..............
      ..............
      ..............
      ..............
  </dsig:Signature>
</soap:Header>
  <soap:Body>
    <product xmlns="http://www.company.com">
      <name>SOA Product</name>
      <company>Company</company>
      <description>WebServices Security</description>
    </product>
  </soap:Body>
</soap:Envelope>
```

The following XPath expression evaluates to true if the <*company*> element contains the value *Company*:

XPath Expression: //ns:company[text()='Company']

Since the <*company*> element actually belongs to the default (xmlns) namespace, i.e. *http://www.company.com*, it is necessary to make up an arbitrary prefix, *ns*, for use in the XPath expression and assign it to *http://www.company.com*. This is necessary to distinguish between potentially several default namespaces which may exist throughout the XML message. The following mapping illustrates this:

Prefix	***URI***
ca	http://www.ca.com

Returning a NodeSet:

Both of the examples above dealt with cases where the XPath expression evaluated to a Boolean value. For example, the expression in the above example asks, "Does the <company> element in the http://www.ca.com namespace contain a text node with the value 'ca'?".

It is sometimes necessary to use the XPath expression to return a subset of the XML message. For example, when using an XPath expression to determine what nodes should be signed in a signed XML message, or when retrieving the nodeset to validate against an XML Schema.

The SOA Security Gateway ships with such an XPath expression: one that returns "All Elements inside SOAP Body". To view this expression, select it from the **Name** field. It appears as follows:

XPath Expression: /soap:Envelope/soap:Body//*

This XPath expression simply returns all child elements of the SOAP <Body> element. To construct and test more complicateD expressions, administrators are advised to use the **XPath Wizard**.

2. XPath Wizard

The **XPath Wizard** assists administrators in creating correct and accurate XPath expressions. The wizard allows administrators to load an XML message and then run an XPath expression on it to determine what nodes are returned. To launch the **XPath Wizard**, click on the **XPath Wizard Button** on the **XPath Expression** dialog.

To use the XPath Wizard, simply enter (or browse to) the location of an XML file in the **File** field. The contents of the XML file will appear in the main window of the wizard. Enter an XPath expression in the **XPath** field and click the **Evaluate** button to run the XPath against the contents of the file. If the XPath expression returns any elements (or

returns true), those elements will be highlighted in the main window.

If you are not sure how to write the XPath expression yourself, you can simply select an element in the main window. An XPath expression to isolate this element is automatically generated and displayed in the **Selected** field. If you wish to use this expression, select the **Use this path** button, and click **OK**.

References

XPath W3C Recommendation [http://www.w3.org/TR/xpath]

Namespaces in XML [http://www.w3.org/TR/REC-xml-names/]

MIME/DIME Settings

Overview

The **MIME/DIME Settings** dialog lists a number of default common content types that are used when transmitting MIME messages. The SOA Security Gateway's **Content Type** filter can be configured to accept or block messages containing specific MIME types. Therefore, the contents of the MIME types library acts as the set of all MIME types that the SOA Security Gateway can filter messages with.

All of the MIME types listed in the table are available for selection in the **Content Type** filter. For example, it is possible to configure this filter to only accept XML-based types - e.g. "application/xml", "application/*+xml", "text/xml", and so on. Similarly, it is possible to block certain MIME types, for example, "application/zip", "application/octet-stream", and "video/mpeg". For more information on configuring this filter, please refer to the Content Type filter help page.

Configuration

The **MIME/DIME Settings** dialog is available by right-clicking on the top-level Policy Store node in the tree view of the SOA Security Gateway Management Console and selecting the **MIME/DIME** menu option.

The **MIME/DIME Settings** dialog lists the actual MIME types on the left column of the table, together with their corresponding file extensions (where applicable) on the right column.

To add a new MIME type, click the **Add** button. In the **Configure MIME/DIME Type** dialog, enter the new content type in the **MIME or DIME Type** field. If the new type has a corresponding file extension, enter this extension in the **Extension** field. Click the **OK** button when finished.

Similarly, existing types can be edited and removed by clicking on the **Edit** and **Delete** buttons, respectively.

Configuring URL Groups

Configuration

The SOA Security Gateway can make connections on a round-robin basis to the URLs listed in a URL group, thus enabling a high degree of failover to external servers.

The SOA Security Gateway will attempt to connect to the listed servers according to the priorities assigned to them. So, for example, let's assume there are two "High" priority URLs, one "Medium" URL, and a single "Low" URL configured. Assuming the SOA Security Gateway can successfully connect to the two "High" priority URLs, it will alternate requests between these two URLs only in a round-robin fashion. The other group URLs will not be used at all. If, however, both of the "High" priority URLs become unavailable, the SOA Security Gateway will then try to use the "Medium" priority URL, and only if this fails will the "Low" priority URL be used.

So, in general, the SOA Security Gateway will attempt to round-robin requests over URLs of the same priority, but will use higher priority URLs before lower priority ones. When a new URL is added to the group it is automatically given the highest priority. Priorities can then be changed by selecting the URL and clicking the **Up** and **Down** buttons.

Individual URLs can be added and edited by selecting the URL from the table and clicking on the **Add** and **Edit** buttons respectively.

The following fields should be completed:

- **URL:**

  Enter the full URL of the external server.

- **Timeout:**

  Specify the timeout in seconds for connections to the specified server.

- **Retry After:**

  Whenever the server becomes unavailable for whatever reason (maintenance, for example), no attempt will be made to connect to that server until the time specified here has elapsed. In other words, once a connection failure has been detected, the next connection to that URL will be made after this amount of time.

- **SSL Certificate:**

If the specified server requires clients to authenticate to it over 2-way SSL, an **SSL Certificate** must be selected from the Trusted Certificate Store for authentication.

- **Host/IP:**

  If the specified server sits behind a proxy server, the host name or IP address of the proxy server must be entered here.

- **Port:**

  Enter the port on which the proxy is listening.

Authentication Repository

Overview

SOA Security Gateway supports a wide range of common authentication schemes, including SSL, XML Signatures, WS-Security Username tokens, and HTTP Authentication. With SSL, the client authenticates to SOA Security Gateway using a client certificate. With XML Signatures, the client is authenticated by validating the signature contained within the XML message. However, when SOA Security Gateway attempts to authenticate a client using a username and password (e.g. WS-Security Username tokens and HTTP Authentication), it must compare the username and password presented by the client to those stored in the **Authentication Repository**.

The **Authentication Repository** acts as a repository for **Users**. **Users** serve many roles in SOA Security Gateway. For example, they can be used to carry out certain privileged tasks, such as generating reports, remote logging, and signing SAML assertions. Similarly, clients whose username and password combinations are stored in the **Authentication Repository** can authenticate to SOA Security Gateway using that username and password combination. For more information on **Users**, take a look at the tutorial here.

The **Authentication Repository** can be maintained in the SOA Security Gateway's local configuration store, a database, or against CA's SOA Security Manager. When a user has been successfully authenticated against one of these repositories, SOA Security Gateway can use any one of that user's stored attributes (e.g. DName, email address, username) to authorize that same user in a subsequent Authorization Filter.

This *credential mapping* is useful, for example, in cases where your client-base uses username-password combinations for authentication (i.e. *authentication attributes*), yet their access rights must be looked up in an authorization server using the client's DName (i.e. *authorization attribute*). In this way, the client possesses a single *virtual identity* within SOA Security Gateway - the client can use one "identity" for authentication, and another for authorization, yet SOA Security Gateway sees both "identities" as representing the same client.

Local Repository Authentication

The **Authentication Repository** can be maintained in the same database as SOA Security Gateway uses to store all its configuration information. Select "Local Repositories" from the **Repository Type** field, and then enter a name for this type of store in the **Repository Name** field.

The **Authorization Attribute Format** field allows administrators to specify whether they want to use the client's **X.509 Distinguished Name** or **User Name** in sub-

sequent Authorization Filters. If "User Name" is selected, the user name used by the client to authenticate to SOA Security Gateway will be used in any configured Authorization Filters. If, on the other hand, "X.509 Distinguished Name" is selected, the X.509 DName stored by SOA Security Gateway for that user will be used for subsequent authorization.

If, for example, the administrator selected the "User Name" option from **Authorization Attribute Format** dropdown, then "admin" (i.e. the **User Name** field) will be used for authorization.

Alternatively, if the "X.509 Distinguished Name" menu option was selected, then the X.509 DName (e.g. "O=Company, OU=comp, EMAIL=emp@company.com, CN=emp") will be used for authorization.

For more information on adding and configuring users to the **Authentication Repository**, take a look at the Users tutorial.

Database Repository Authentication

SOA Security Gateway can store its **Authentication Repository** in an external database. This option makes sense when an organization already has a "silo" of user profiles stored in the database and does not want to duplicate this store within SOA Security Gateway's local configuration storage.

To authenticate users against a database repository, complete the following fields:

Repository Type and Name

Select "Database Repository" from the **Repository Type** field, and enter an appropriate name for the database in the **Repository Name** field.

Database

There are 2 basic configuration items required to retrieve a user's profile from the database: the location of the database and the query to use to retrieve the appropriate user's profile.

- **Database Location:**

  Connection details for the database can be configured by clicking on the **Add** button and completing the **Database Connection** dialog. For more information on configuring the fields on this dialog, please refer to the Database Connection guide.

  Previously configured database connections can be edited and removed by selecting them in the dropdown and clicking the **Edit** and **Delete** buttons respectively.

- **Database Query:**

  The **Database Query** is used to retrieve a specific user's profile from the database so that SOA Security Gateway can then authenticate them. Having successfully authenticated the user, it is possible to select an attribute of this user to use for authorization filter later on in the policy. The **Database Query** can take the form of a SQL statement, stored procedure, or function call. For more information on how to configure the **Database Query**, please refer to the Database Query Configuration guide.

Format Password Received From Client:

If the user sends up a clear-text password to SOA Security Gateway, but that user's password is stored in a hashed format in the database, then it is necessary for SOA Security Gateway to hash the password before performing the authentication step.

- **Hash Client Password:**

  Depending on whether you wish to hash the user's submitted password or not, select the appropriate radio button.

- **Hash Format:**

  If you have selected to hash the client's password, SOA Security Gateway needs to know the format of the hashed password. The most typical formats are available from the dropdown, however, it is also possible to enter another format. Formats should be entered in terms of message attributes. The following 2 formats are available from the **Hash Format** dropdown. The first option combines the username, authentication realm, and password respectively. This combination is then hashed. The second option simply creates a hash of the user's password.

${authentication.subject.id}:${authentication.subject.realm}:${authentication.subject.password}

${authentication.subject.password}

- **Hash Algorithm:**

Select either **MD5** or **SHA1** to use as the digest algorithm to use when creating the hash.

Query Result Processing

This section of the screen allows you to provide SOA Security Gateway with some meta information about the result returned by the **Database Query** configured earlier on this screen. Specifically, it allows you to identify the name of the database table column or row that contains the user's password, and also the name of the column or row that contains the attribute that is to be used for the authorization filter.

- **Password Column:**

  Specify the name of the database table column that contains the user's password. The contents of this column will be compared to the password submitted by the user.

- **Password Type:**

  Depending on how the user's password has been stored in the database, select either "Clear Password" or "Digest Password" from the dropdown.

- **Authorization Attribute Column:**

  By running the **Database Query**, all of the user's attributes are returned. Only the user's username and password are used for the actual authentication event. It is also possible to use one of the other user's attributes for authorization at a later stage in the policy. The additional "authorization attribute" should be either a username or an X.509 distinguished name (DName). The name of the column containing either the username or the Dname should be entered here, but only if this value is required for authorization purposes.

- **Authorization Attribute Format:**

  SOA Security Gateway's authorization filters all operate on the basis of a username or DName. In other words, they all evaluate whether a user identified by a username or DName is allowed to access a specific resource. Select the appropriate format from the dropdown depending on what type of user credential is stored in the database table column entered above.

Authenticate to CA SOA Security Manager

In cases where user profiles have been stored in an existing CA SOA Security Manager, SOA Security Gateway can query the SOA Security Manager in order to authenticate end-users.

Configure the following fields:

Connection Details:

SOA Security Gateway uses the information in the Security Manager hosts configuration file in order to connect to CA SOA Security Manager. This file is called "SmHost.conf" by default. Browse to the location of this file using the button at the bottom right-hand corner of the **Connection Details** text area. After selecting the configuration file, the connection details will appear in the text area.

Agent Name:

Enter the name of the agent as configured with the SOA Security Manager.

Resource:

Enter the name of the protected resource for which the end-user must be authenticated.

It is possible to enter wildcards representing message attributes in this field. Wildcards have the following format:

```
${message.attribute}<br/>
```

So, for example, to specify the original path on which the request was received by SOA Security Gateway as the resource, enter the following wildcard:

```
${http.request.uri}
```

Action

The end-user must be authenticated for a specific action on the protected resource. By default this action is taken from the HTTP verb used in the incoming request. The following wildcard is used to get the HTTP verb:

```
${http.request.verb}<br/>
```

Alternatively, any user-specified value can be entered here.

Create Single Sign-On Token

When this option is selected, CA SOA Security Manager will generate a single sign-on token as part of the authentication event and return it to SOA Security Gateway. It will then be inserted into the downstream message for re-use at a later stage, either by another instance of SOA Security Gateway running the **SiteMinder Session Validation** filter, or by another SiteMinder-aware agent.

Put Token in Message Attribute

Enter the name of the message attribute where you wish to store the single sign-on token. By default, the token will be stored in the `siteminder.session` attribute. Please refer to the Message Attribute Reference for a complete list of available message attributes.

LDAP User Search

Configure Directory Search

The **User Search** dialog is used to search a given LDAP directory for a unique user according to the criteria configured in the fields on this dialog.

Base Criteria:

The value entered here tells the SOA Security Gateway where it should begin searching the LDAP directory. For example, it may be appropriate to search for a given user under the "C=IE" tree in the LDAP hierarchy.

Query Search Filter:

The value entered here is what the SOA Security Gateway will use to determine whether it has obtained a successful match or not. In this case, since we are searching for a specific user, we can use the username of an authenticated user (i.e. the value of the `authentication.subject.id` message attribute to lookup in the LDAP directory. We must also specify the object class that defines users for the particular type of LDAP directory that we are searching against. For example, object classes representing users amongst common LDAP directories are "inetOrgPerson", "givenName", and "User".

So, for example, to search for an authenticated user against Microsoft's Active Directory, you might specify the following as the **Query Seach Filter**:

```
(objectclass=User)(cn=${authentication.subject.id})
```

Search Scope:

The checkboxes here indicate the depth of the LDAP tree that you wish to search. The choice selected here will depend largely on the structure of your LDAP directory.

Setting the Encryption Passphrase

Encryption Passphrase Overview

By default, data is stored unencrypted in the Policy Store. It is, however, possible to encrypt certain sensitive information, such as passwords and private keys, using a passphrase. Once the passphrase has been set (and Policy Store data has been encrypted with it), it must be entered when connecting to the Policy Store with both the SOA Security Gateway Management Console and the SOA Security Gateway so that these components can decrypt the encrypted data.

This help page describes how to set the passphrase for the first time and then how to specify this passphrase when connecting to the Policy Store with both the SOA Security Gateway Management Console and the SOA Security Gateway. We also describe how to change the passphrase once it has been set initially.

Setting the Passphrase for the First Time

Complete the following steps to set the encryption passphrase for the first time:

1. Open up a command prompt at the `INSTALL_DIR/PLATFORM/bin` directory, where "INSTALL_DIR" points to the root of your product installation and "PLATFORM" corresponds to the platform on which you have installed the product. Start the SOA Security Gateway using the `soagateway` startup script.
2. Start the SOA Security Gateway Management Console and connect to the management interface exposed by the SOA Security Gateway, which is available by default at

   ```
   http://HOST:8090/configuration/policies
   ```

   where "HOST" refers to the host or IP address of the machine on which the SOA Security Gateway was installed.
3. Right-click on the top level node in the tree view of the SOA Security Gateway Management Console, which represents the underlying Policy Store where all configuration data is stored. Select the **Change Passphrase** option from the context menu.
4. Enter the passphrase that you want to use to encrypt sensitive data in the Policy Store in the **New Passphrase** field. Make sure to leave the **Old Passphrase** field blank if you are setting the passphrase for the first time.
5. The new passphrase must now be specified when connecting to the Policy Store

with the SOA Security Gateway Management Console and also with the SOA Security Gateway. The following instructions describe how to do this for both components.

Connecting to the Policy Store with the SOA Security Gateway Management Console

Having set the encryption passphrase for the Policy Store, this passphrase must now be specified every time you connect to the Policy Store (via the SOA Security Gateway's management interface) with the SOA Security Gateway Management Console.

The passphrase can be entered in the **Passphrase Key** field of the **Connection Details** dialog, which is displayed when the SOA Security Gateway Management Console is starting up.

It is important to note the different roles of the **Passphrase Key** and **Password** values that are specified on this screen:

Passphrase Key:

The **Passphrase Key** entered here is used to decrypt sensitive data (e.g. private keys) that have already been encrypted in the Policy Store. It is *not* required by default and is only needed if the steps outlined above have been carried out.

Password:

On the other hand, the **Password** specified here is used to authenticate to the SOA Security Gateway's management interface using HTTP basic authentication. It *is* required by default.

Connecting to the Policy Store with the SOA Security Gateway

In order for the SOA Security Gateway to read (i.e. decrypt) encrypted data from the Policy Store, it must be primed with the passphrase key. This can be done either by specifying the passphrase directly in the SOA Security Gateway configuration file, or by prompting for it when the SOA Security Gateway is starting up:

Specifying the Passphrase in the Configuration File:

It is possible to specify the password directly in the SOA Security Gateway's configuration file. To do this, open the `userconfig.dtd` file, which can be found in the `/conf` directory of your product installation. This DTD file contains values for general system settings, including the username and password to use to authenticate to the management interface exposed by the SOA Security Gateway and also (if required) the passphrase key to use to decrypt encrypted data stored in the Policy Store.

Typically the password is only entered directly in the file if the SOA Security Gateway must be started as an NT Service or UNIX daemon. In this case, it is not possible for an administrator to enter a password manually when the SOA Security Gateway is starting. To avoid this problem, the password must be entered in the configuration file. To do this, specify the passphrase as the value for the `server.entitystore.secret` entity as follows, where "myPassphrase" is the encryption passphrase:

```
<!ENTITY server.entitystore.secret      "myPassphrase">
```

Prompting for the Passphrase on Server Startup:

If you would rather not specify the passphrase in the configuration file and do not need to start the SOA Security Gateway as an NT Service or UNIX daemon, you can configure the SOA Security Gateway to prompt the administrator for the passphrase when starting up. To do this, enter the special value "(prompt)" as the value of the `server.entitystore.secret` entity as follows:

```
<!ENTITY server.entitystore.secret      "(prompt)">
```

It is important to note that if you use this option, you must take care to remember the encryption passphrase. Failure to use the correct passphrase will result in loss of private key data and may prevent the SOA Security Gateway from functioning correctly.

Changing the Passphrase

If you have already specified a passphrase to use to encrypt the data, you can change this passphrase by selecting the **Change Passphrase** option from the Policy Store context menu. Complete the following fields on the **Change Encryption Passphrase** dialog:

Old Passphrase:

Enter the old passphrase that you wish to change in this field.

New Passphrase:

Enter the new passphrase here.

Old Passphrase:

Confirm the new passphrase in this field.

Retrieving WSDL Files from a UDDI Directory

Overview

A WSDL (Web Services Description Language) file defines the interface to a Web Service, or list of services. It lists the operations exposed by the service, the wire format for requests for these operations, and the data types of any elements that appear within the body of the requests. It also gives the location of the Web Service in terms of a URL that clients can access in order to use the exposed operations.

The SOA Security Gateway Management Console can extract this information from the WSDL file to generate the following filters, which can then be incorporated into one or more policies:

- **Relative Path Resolver**
- **SOAPAction Resolver**
- **SOAP Operation Resolver**
- **Connection Handler**
- **Static Router**
- **Schema Validation**

WSDL files are used to create Web Services in the **Web Services Repository** and also when adding XML Schemas to the global **Schema Cache**. In both cases, a WSDL file can be retrieved from the file system, from a URL, or from a UDDI registry. The remainder of this help page explains how to retrieve a WSDL from a UDDI registry.

UDDI: A Brief Introduction

UDDI (Universal Description, Discovery and Integration) is an OASIS-led initiative that allows businesses to publish and discover services on the public Internet. A business publishes services that it provides to a global XML-based registry in such a way that other businesses can dynamically look up the registry and discover these services. Enough information is published to the registry to allow other businesses to find services and communicate with them.

There are 3 aspects to a business registration in a UDDI registry:

- **Green Pages:**

Contains technical information about the services exposed by the business

- **Yellow Pages:**

  Categorizes the services according to standard taxonomies and categorization systems

- **White Pages:**

  Gives general information about the business, such as name, address, and contact information

Of key importance to the SOA Security Gateway Management Console is that the UDDI registry can be searched according to a whole range of search criteria. The object of this exercise is to retrieve the WSDL file for a particular service. The SOA Security Gateway Management Console can then use this WSDL file to create a policy for the service and/or to extract a schema from the WSDL to check the format of messages attempting to use the operations exposed by the Web Service.

For a more detailed description of UDDI, the reader is advised to consult the UDDI specification. In the meantime, the next section gives some high-level definitions of some of the terms that appear in the SOA Security Gateway Management Console interface.

UDDI Definitions

Since UDDI terminology is used throughout the screens that comprise the **Load WSDL** wizard (and therefore, throughout this help page), the following list of definitions explain some common UDDI terms. For more detailed explanations of these terms, please refer to the UDDI specification.

businessEntity:

This represents all known information about a business, such as name, description, contact information, and so on. A businessEntity may have an identifierBag, which is a list of name-value pairs used to hold identifiers, such as DUNS (Data Universal Numbering System) numbers or taxonomy identifiers. A businessEntity may also have a categoryBag, which is a list of name-value pairs used to tag the businessEntity with taxonomy/classification information such as industry, product,or geographic codes. Furthermore, a businessEntity may contain a number of businessService entities.

businessService:

A businessService represents a single logical service classification. It is used to describe a service provided by a business. It contains descriptive information in business terms outlining the type of technical services found within each businessService element. A

businessService may have a categoryBag and may contain a number of bindingTemplate entities.

bindingTemplate:

A bindingTemplate contains pointers to the technical descriptions and the access point URL of the service, but does not contain the details of the service's specification. A businessService may contain references to a number of tModel entities.

tModel:

A tModel is keyed metadata that is used in UDDI as a keyed namespace reference. A tModel consists of a key, a name, a description, and a URL. tModels are referred to by other entities in the registry. The tModel's primary role is to represent a technical specification. A specification designer can establish a unique technical identity within a UDDI registry by registering information about the specification in a tModel. Other parties can express the availability of Web services that are compliant with a specification by including a reference to the tModel in their bindingTemplate data.

This approach facilitates searching for registered Web services that are compatible with a particular specification. tModels are also used within identifierBag and categoryBag structures to define organizational identity and various classifications. Used in this context, the tModel reference represents a relationship between the keyed name-value pairs to the super-name, or namespace within which the name-value pairs are meaningful. A tModel may have an identifierBag and a categoryBag.

Identification:

The purpose of identifiers in a UDDI registry is to allow others to find the published information using more formal identifiers such as DUNS numbers, Global Location Numbers (GLN), tax identifiers, or any other kind of organizational identifiers, regardless of whether these are private or shared.

The following are identification systems used commonly in UDDI registries:

Name:	*Description:*	*tModel UUID:*
dnb-com:D-U-N-S	Dun and Bradstreet D-U-N-S Number	uuid:8609C81E-EE1F-4D5A-B202-3EB13AD01823
thomasregister-com:supplierID	Thomas Registry Suppliers	uuid:B1B1BAF5-2329-43E6-AE13-BA8E97195039

Categorization:

Entities in the registry may be categorized according to categorization system defined in a tModel, for example, geographical region. The businessEntity, businessService, and tModel types have an optional categoryBag. This is a collection of categories each of which have a name, value, and tModel key.

The following are categorization systems used commonly in UDDI registries:-

Name:	*Description:*	*tModel UUID:*
uddi-org:types	UDDI Type Taxonomy	uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4
ntis-gov:naics:1997	North American Industry Classification System (NAICS) 1997 Release	uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2

Registry Configuration

We first need to select the UDDI registry that we want to search for WSDL files. Complete the following fields to select or add a UDDI registry:

Registry:

Select an existing UDDI registry to browse for WSDL files from the **Registry** dropdown. To configure the location of a new UDDI registry, click the **Add** button. Similarly, to edit an existing UDDI registry location click the **Edit** button. The **Registry Connection Details** dialog is displayed. Complete the following fields:

Registry Name:

Enter display name for the UDDI registry in this field.

Inquiry URL:

Enter the Inquiry URL of the UDDI registry.

Registry Authentication Type:

This field is optional. The only supported authentication type is "UDDI_GET_AUTHTOKEN".

Username:

Enter the username required to authenticate to the registry, if required.

Password:

Enter the password for this user, if required.

Proxy Host:

If the UDDI registry location entered above requires a connection to be made through a HTTP proxy enter the host name of the proxy here.

Proxy Port:

If a proxy is required enter the port on which the proxy server is listening.

Username:

If the proxy has been configured to only accept authenticated requests, the SOA Security Gateway Management Console will send this username and password to the proxy using HTTP basic authentication.

Password:

Enter the password to use along with the username specified in the field above.

SSL Proxy Host:

If the **Inquiry URL** specified above uses the HTTPS protocol, the SSL proxy host entered here will be used instead of the HTTP proxy entered above. In this case the proxy settings entered above will not be used.

SSL Proxy Port:

Enter the port that the SSL proxy is listening on.

Quick Search

The **Quick Search** facility allows you to search the UDDI registry for all tModels that have been categorized according to the uddi-org:types categorization system as "wsdlSpec". The user may optionally enter a tModel name in the **Name** field in order to fine-grain the search.

The name entered is a partial or full name pattern with wildcard searching as specified by the *SQL-92 LIKE* specification . The wildcard characters are percent '%', and underscore '_', where an underscore matches any single character and a percent matches zero or more characters.

Click the **Search** button to initiate the search. The **Search Results** tree shows the tModel URIs as top-level nodes. These URIs are all WSDL URIs and can be used to generate policies by selecting the URI and clicking the **Next** button.

The user may click on any of the nodes in the tree to display detailed properties about that node in the table below the **Search Results** tree. The properties listed will depend on the type of the node that is selected.

Search by Name

The **Name Search** allows you to search for tModels by name, but also to specify what entity level in the tree to start the search from. The user must select one of the following entity levels to start the search from:

- **businessEntity**
- **businessService**
- **tModel**

Click the **Search** button to start the name search. The search results will display the matching entities from the starting level down to the tModel level. For example, if the user selects the **businessService** option, the search results will show businessServices as top level nodes, then bindingTemplates, then tModels.

The user may optionally enter a name in the **Name** field in order to narrow the search. Wildcards may be used for the name. The name will apply to a businessEntity, businessService, or tModel depending on which registry entity type has been selected. If no name is entered, all entities of the selected type are retrieved.

It is important to note that the tModel URIs shown in the resulting tree may not all be categorized as "wsdlSpec" according to the uddi-org:types categorization system. The user may choose to load any of these URIs as a WSDL file, but they will be warned if it is not categorized as "wsdlSpec".

As before, the user may click on any node in the results tree to display properties about that node in the table.

Advanced Search

With the **Advanced Search** tab, the user can search the UDDI registry using any combination of **Names**, **Keys**, **tModels**, **Discovery URLs**, **Categories**, and **Identifiers**. Furthermore, it is also possible to specify the entity level in the tree to start searching from. All of these options combine to provide a very powerful search facility.

It is possible to specify search criteria for any of the categories listed above by right-

clicking on the folder node in the **Enter Search Criteria** tree and selecting the **Add** menu option. It is possible to enter more than 1 search criteria of the same type, e.g. 2 search criteria of type Keys.

It is important to note that the tModel URIs shown in the resulting tree may not all be categorized as "wsdlSpec" according to the uddi-org:types categorization system. The user may choose to load any of these URIs as a WSDL file, but they will be warned if it is not categorized as "wsdlSpec".

The following list explains how to add a search criteria for each of the types listed in the **Enter Search Criteria** tree. All search criteria are configured by right-clicking on the folder node and selecting the **Add** menu option.

Names:

Simply enter a name to be used in the search in the **Name** field on the **Name Search Criterion** dialog. For example, the name could be the name of a businessEntity.

As with all name searches, the name is a partial or full name pattern with wildcards allowed as specified by the *SQL-92 LIKE* specification. The wild-card characters are percent '%', and underscore '_', where an underscore matches any single character and a percent matches zero or more characters.

A name search criterion may be used for businessEntity, businessService, and tModel level searches.

Keys:

On the **Key Search Criterion** dialog, you can specify a key to search the registry for in the **Key** field. The key value is a UUID (Universally Unique Identifier) value for a registry object. The **Key Search Criterion** can be used on all levels of searches.

If 1 or more keys are specified with no other search criteria, the keys are interpreted as the keys of the selected type of registry object and used for a direct lookup, as opposed to a find/search operation. For example, if the user enters "key1" and "key2" and selects the businessService entity type, the search will retrieve the businessService object with key "key1", and another businessService with key "key2".

If a key is entered with other search criteria, then a key criterion will be interpreted as follows:

- For a businessService entity lookup, the key will be the businessKey of the services
- For a bindingTemplate entity lookup, the key will be the serviceKey of the binding templates
- Not applicable for any other object type

tModels:

The user can enter a key in the **tModel Key** field on the **tModel Search Criterion** screen. The key entered should correspond to the UUID of the tModel associated with the type of object we are searching for.

A tModel search criterion may be used for businessEntity, businessService, and bindingTemplate level searches.

Discovery URLs:

Enter a URL in the **Discovery URL** field on the **Discovery URL Search Criterion** dialog. The **Use Type** field is optional, but can be used to further fine-grain the search by type.

A Discovery URL search criterion may be used for businessEntity level searches only.

Categories:

The user must select a previously configured categorization system from the **Type** dropdown on the **Category Search Criterion** dialog. The dropdown is pre-populated with a list of common categorization systems. A new categorization system can be added by clicking the **Add** button.

On the **Add/Edit Category** dialog, enter a **Name**, **Description**, and **UUID** for the new category type in the fields provided.

Once the categorization system has been selected or added, the user must enter a value to search for in the **Value** field. The **Name** field is optional.

Identifiers:

A previously configured identification system must be selected from the **Type** dropdown on the **Identifier Search Criterion** dialog. The content of this dropdown is pre-populated with well-known identification systems. To add a new identification system, click the **Add** button.

On the **Add/Edit Identifier** dialog, enter a **Name**, **Description**, and **UUID** for the new identifier in the fields provided.

Options

This tab allows you to configure various aspects of the search conditions specified on the previous tabs. The following options are available:

Exact Match:

If this checkbox is checked, the name entered as part of the search criteria must exactly match the name specified in the UDDI registry.

Case Sensitive:

Determines whether the name entered by the user must match the case of that stored in the registry.

Sort by Ascending Name:

Sorts the results alphabetically in order of ascending name.

Sort by Descending Name:

Sorts the results alphabetically in order of descending name.

AND all Keys:

Identifier search criteria will be ORed together by default. This setting will ensure they are ANDed instead.

OR all Keys:

tModel and category search criteria are ANDed by default. The selection of this setting will OR these criteria instead.

OR like Keys:

When a bag container contains multiple keyedReference elements (i.e., categoryBag or identifierBag), any keyedReference filters that come from the same namespace (e.g. have the same tModelKey value) are ORed together rather than ANDed. This allows one to say, "any of these four values from this namespace, and any of these two values from this namespace".

Combine Category Bags:

This qualifier makes the categoryBag entries of a businessEntity behave as though all categoryBags found at the businessEntity level and in all contained or referenced businessServices were combined. Searching for a category will yield a positive match on a registered business if any of the categoryBags contained within a businessEntity (including the categoryBags within contained or referenced businessServices) contains the filter criteria.

Service Subset:

This qualifier causes the component of the search that involves categorization to use only the categoryBags from *directly* contained or referenced businessServices within the registered data. The search results will return only those businesses that matched based on this modified behavior, in conjunction with any other search arguments provided.

References

UDDI [http://www.uddi.org]

WSDL [http://www.w3.org/TR/wsdl]

Default Settings

Overview

The **Default Settings** interface allows the administrator to set several global configuration settings in order to tweak the behavior of the SOA Security Gateway.

These settings can be overwritten at the Process level by right-clicking on the Process node in the tree and selecting the **Settings -> Custom** menu option.

To configure the **Default Settings**, right-click on the top-level Policy Store node in the SOA Security Gateway Management Console, and select the **Default Settings** menu option, which displays the . **Default Settings** dialog.

Settings

Setting	***Purpose***
Date Format	Configures the format of the date for the purposes of tracing, logging, and reporting. For more information, please refer to http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html.
Cache Refresh Interval	This setting configures the number of seconds that the server will cache data loaded from an external source (e.g. external database, LDAP directory, etc) before refreshing the data from that source. The default value is 5 seconds.
LDAP Service Provider	This setting denotes the service provider used for looking up an LDAP server e.g. com.sun.jndi.ldap.LdapCtxFactory. The provider is typically used to connect to LDAP directories for certificate and attribute retrieval.
Realm	Specifies the realm for authentication purposes.
Schema Pool Size	Sets the size of the Schema Parser pool.
Server Brand	Specifies the branding to be used in the product.
LDAP Time Out	This is the timeout in milliseconds for the

Setting	***Purpose***
	LDAP connection. If a connection has not been created within this timeframe the operation will fail will a timeout. Similarly if a lookup operation has not succeeded within this timeframe, it will fail. If this setting is not configured, or set to zero, the TCP timeout for the platform is used, which defaults to 3 minutes.
Token Drift Time	The number of seconds drift allowed for WS-Security tokens. This is important in cases where the SOA Security Gateway is checking the date on ingress WS-Security tokens. It is likely that the machine on which the token was created will be out-of-sync with the machine on which the SOA Security Gateway is running. The drift time allows for differences in the respective machine clock times.
Use Validation on SAX Parsers	This setting is disabled by default for performance reasons. It can be enabled, however, to perform SAX validation when parsing XML messages.
Idle Timeout	The SOA Security Gateway supports the use of HTTP 1.1 persistent connections. The **Idle Timeout** is the time that the SOA Security Gateway will wait after sending a message over a persistent connection before it closes the connection. Typically, the host will tell the SOA Security Gateway that it wants to use a persistent connection. The SOA Security Gateway acknowledges this instruction and decides that it will keep the connection open for a certain amount of time after sending the message to the host. If the connection is not reused by within the **Idle Timeout** period, the SOA Security Gateway will close the connection. It is important to note that this setting can be configured on a per-host basis by configuring the **Remote Hosts** interface.

Setting	***Purpose***
Active Timeout	When the SOA Security Gateway receives a large HTTP request, it reads the request off the network as it becomes available. If the time between reading successive blocks of data exceeds the **Active Timeout**, the SOA Security Gateway will close the connection. The idea here is to guard against a host closing the connection while in the middle of sending data. Imagine the host's network connection is pulled out of the machine while in the middle of sending data to the SOA Security Gateway. When the SOA Security Gateway has read all the available data off the network, it will wait the **Active Timeout** period of time before closing the connection. It is important to note that this setting can be configured on a per-host basis by configuring the **Remote Hosts** interface.
Maximum Memory per Request	The maximum amount of memory that will be allocated to each request. It is important to note that this setting can be configured on a per-host basis by configuring the **Remote Hosts** interface.
Tracing Level	Allows you to set the tracing level for the SOA Security Gateway at run-time. Select the appropriate option from the **Trace Level** dropdown.

Namespace Settings

Overview

The SOA Security Gateway exposes a global setting that allows you to configure what versions of the SOAP and WSSE specifications it supports. Furthermore, it allows you to specify what attribute is used to identify the XML Signature reference within a SOAP message.

The **Namespace Settings** configuration screen is available by right-clicking on the top-level Policy Store node in the SOA Security Gateway Management Console, and selecting the **Namespace Settings** menu option. Namespaces are configured using the **Namespace Settings** dialog.

Signature ID Attribute

The **Signature ID Attribute** tab allows you to list the supported attributes that can be used by the SOA Security Gateway to identify a Signature reference within an XML message.

An XML-signature `<signedInfo>` section may reference signed data via the `URI` attribute. The `URI` value may contain an id that identifies data in the message. The referenced data will hold the "URI" field value in one of its attributes.

By default, the server will use the "Id" attribute for each of the WSSE namespaces listed above to locate referenced signed data. The following sample XML Signature illustrates the use of the "Id" attribute:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Header>
  <dsig:Signature id="Sample" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
   <dsig:SignedInfo>
    ...
    <dsig:Reference URI="#CA:sLmDCph3tGZ10">
      ...
    </dsig:Reference>
   </dsig:SignedInfo>
    ....
  </dsig:Signature>
 </soap:Header>
 <soap:Body>
  <getProduct wsu:Id="CA:sLmDCph3tGZ10"
      xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility">
   <Name>SOA Test Client</Name>
   <Company>Company</Company>
  </getProduct>
 </soap:Body>
</soap:Envelope>
```

It is clear from this example that the Signature reference identified by the `URI` attribute of the `<Reference>` element refers to the nodeset identified with the `Id` attribute, i.e. the `<getProduct>` block.

Because different toolkits and implementations of the XML-Signature specification can use attributes other than the `Id` attribute, the SOA Security Gateway allows the user to specify other attributes that should be supported in this manner. By default, the SOA Security Gateway supports the `Id`, `ID`, and `AssertionID` attributes for the purposes of identifying the signed content within an XML Signature.

However it is possible to add more attributes by clicking the **Add** button and adding the attribute in the interface provided. The priorities of attributes can be altered by clicking the **Up** and **Down** buttons. For example, if most of the XML Signatures processed by the SOA Security Gateway use the `ID` attribute, this attribute should be given the highest priority.

WSSE Namespace

The **WSSE Namespace** tab is used to specify the WSSE (and corresponding WSSU) namespaces that are supported by the SOA Security Gateway.

The SOA Security Gateway attempts to identify WS Security blocks belonging to the WSSE namespaces listed in this table. It first attempts to locate Security blocks belonging to the first listed namespace, followed by the second, then the third, and so on until all namespaces have been utilized. If no Security blocks can be found for any of the listed namespaces, the message will be rejected on the grounds that the SOA Security Gateway does not support the namespace specified in the message. To add a new namespace, click the add button.

It is important to note that every WSSE namespace has a corresponding WSSU namespace. For example, the following WSSE and WSSU namespaces are inextricably bound:

WSSE Namespace	http://schemas.xmlsoap.org/ws/2003/06/secext
WSSU Namespace	http://schemas.xmlsoap.org/ws/2003/06/utility

First, enter the WSSE namespace in the **Name** field. Then enter the corresponding WSSU namespace in the **WSSU Namespace** field.

SOAP Namespace

The **SOAP Namespace** tab can be used to configure the SOAP namespaces that are supported by the SOA Security Gateway. In a similar manner to the way in which the SOA Security Gateway handles WSSE namespaces, the SOA Security Gateway will attempt to identify SOAP messages belonging to the listed namespaces in the order given in the table.

The default behavior is to attempt to identify SOAP 1.1 messages first, and for this reason, the SOAP 1.1 namespace is listed first in the table. The SOA Security Gateway will only attempt to identify the message as a SOAP 1.2 message if it can't be categorized as a SOAP 1.1 message first.

References

XML Signature [http://www.w3.org/TR/xmldsig-core/]

Web Services Security [http://docs.oasis-open.org/wss]

SOAP Specifications [http://www.w3.org/TR/soap/]

Find Filter Dialog

Overview

The **Find Filter Dialog** can be used to find a named filter of a particular type. It is is especially useful in cases where many policies have been configured with the SOA Security Gateway Management Console and the administrator wants to find a specfic filter. For example, you could find a schema filter called, "Check against SOAP Schema".

Configuration

The **Find Filter Dialog** is available from the **Edit -> Find Filter** main menu option. Enter the name of the filter to find in the **Name** field, and then select its type from the **Filter Type** dropdown. Click the **Search** button to find the named filter.

If you want to search for all filters of a particular type, select the type from the **Filter Type** dropdown and leave the **Name** field blank. Click the **Search** button to find all filters of this type. The filters will be displayed in the **Search Results** table.

Similarly, to display a listing of all configured filters, simply leave both fields blank and click the **Search** button.

It is possible to right-click on a filter displayed in the **Search Results** table and then select any of the following options from the context menu:

Edit:

Use this option to edit the selected filter.

Delete:

Select this option to delete the selected filter from the policy in which it was configured.

Log Level:

Select the log level(s) that this filter will log at using the relevant menu options. All filters can log at "Failure", "Fatal", and "Success" levels. To edit the log message for this filter, select the **Edit** option from the context menu to display the configuration data for the filter, and then click the **Next** button to edit the log messages.

View Item:

When this menu option is selected, the filter will be displayed in the policy tree-view on

the left hand side of the SOA Security Gateway Management Console. Furthermore, the policy in which this filter runs will be displayed in the policy pallette.

Cryptographic Acceleration

Overview

The SOA Security Gateway uses OpenSSL to perform cryptographic operations, such as encryption and decryption, signature generation and validation, and SSL tunneling. OpenSSL exposes an *Engine API*, which makes it possible to plug in alternative implementations of some or all of the cryptographic operations implemented by OpenSSL. OpenSSL will, if configured appropriately, call the engine's implementation of these operations instead of its own.

So, for example, a particular engine may provide improved implementations of the asymmetric operations RSA and DSA. This engine can then be plugged into OpenSSL so that whenever OpenSSL needs to perform either an RSA or DSA operation, it will call out to the engine's implementation of these algorithms rather than call its own.

Typically, OpenSSL engines provide a hardware implementation of specific cryptographic operations. The hardware implementation usually offers improved performance over its software-based counterpart, i.e. it offers *cryptographic acceleration*.

Cryptographic acceleration can be configured at the process level in the SOA Security Gateway. To configure an SOA Security Gateway process to use an OpenSSL engine instead of the default OpenSSL implementation, right-click on the process in the tree-view in the **SOA Security Gateway Management Console** and select the **Cryptographic Acceleration -> Add OpenSSL Engine** menu option.

General Configuration

The **OpenSSL Engine Configuration dialog:**

The dialog displays the name of the engine, the algorithms that it implements, together with any initialization and cleanup commands required by the engine. Complete the following fields:

Name:

Enter an appropriate name for the engine in this field.

Provides:

Enter a comma-separated list of cryptographic operations that are to be performed by the engine as opposed to OpenSSL. The engine must implement the listed operations, otherwise the default OpenSSL operations will be used. The following operations are available:

RSA	The RSA (Rivest Shamir Adleman) asymmetric algorithm
DSA	The DSA (Digital Signature Algorithm) asymmetric algorithm
RAND	Random number generation
DH	The Diffie-Hellman anonymous key exchange algorithm
ALL	Use the engine's implementation of **all** cryptographic algorithms

So, for example, if you want to configure the SOA Security Gateway to use the engine's implementation of the RSA, DSA, and DH algorithms only, enter the following in the **Provides** field:

```
RSA, DSA, DH
```

Commands:

The OpenSSL engine framework allows a number of control commands to be invoked at various stages in the loading and unloading of a specific engine library. These commands can be issued before and/or after the initialization of the engine and also before and/or after the engine is un-initialized. Control commands are based on textual name-value pairs.

Typical uses for control commands include specifying the path to a driver library, logging configuration information, a password to access a protected devices, a configuration file required by the engine, and so on.

OpenSSL control commands can be added by clicking the **Add** button. The **OpenSSL Engine Command**:

Enter the name of the command in the **Name** field, and its value in the **Value** field. This command ***must*** be supported by the engine.

Use the **When** dropdown to select when the above command is to be run. The options available are as follows:

preInit	Commands will be run **before** the engine is initialized, i.e. **before** the call to `ENGINE_init()`.
postInit	Commands will be run **after** the engine is initialized, i.e. **after** the call to `ENGINE_init()`.
preShutdown:	Commands will be run **before** the engine shuts down, i.e. **before** the call to `ENGINE_finish()`.
postShutdown	Commands will be run **after** the engine shuts down, i.e. **after** the call to `ENGINE_finish()`.

Connection Details

Overview

The **Connection Details** page allows the user to connect to different types of Policy Stores. The Policy Store, which is responsible for storing all configuration data for the SOA Security Gateway, is inherently storage-agnostic and can, therefore, be stored be stored in any of the supported types of data store:

1. CA Server
2. File System
3. Database
4. Directory Server
5. XML Database

Select the appropriate option and click the **Next** button to configure the fields necessary to connect to this type of Policy Store. By default, the SOA Security Gateway Management Console connects to the Management Service exposed by a running instance of the SOA Security Gateway and as such, the **CA Server** radio button is selected.

The following sections describe how to connect to each of the various types of Policy Store supported by the SOA Security Gateway.

CA Server

The SOA Security Gateway exposes a Management Service interface to its underlying Policy Store. This enables SOA Security Gateway Management Consoles running on different machines to that on which the SOA Security Gateway is installed to administer policies remotely.

To connect to the Management Service interface of a running instance of the SOA Security Gateway, configure the following fields:

Server URL:

Enter the URL of the Management Service interface exposed by the SOA Security Gateway. By default, this interface is available at `http://HOST:8090/configuration/policies`, where "HOST" points to the IP address or host name of the machine on which the SOA Security Gateway is running.

User Name:

The Management Service interface is protected by HTTP basic authentication. Therefore, a username and password must be provided so that the SOA Security Gateway Management Console can authenticate to the SOA Security Gateway. By default, the SOA Security Gateway User Store contains a user "admin" with password "changeme", which can be used in this case. It is possible to change this user's details using the User Store interface.

Password:

Specify the password for the user in this field. The password for the default "admin" user is "changeme".

Passphrase Key:

All sensitive data (password, keys etc.) stored in the Policy Store can be encrypted using a passphrase. If you wish to do this you can enter a password in this field when connecting to the Management Service interface. You must use this password thereafter when connecting to this interface.

It is important to note that if you have chosen to use a passphrase here, it is crucial that you do not forget this password. Failure to enter the correct passphrase will result in loss of private key data.

File System

Because the Policy Store is stored in an XML file by default, the SOA Security Gateway Management Console can be configured to connect directly to the Policy Store file. To do this, complete the following fields:

File:

Enter or browse to the location of the Policy Store file using the fields provided.

Passphrase Key:

All sensitive data (password, keys etc.) stored in the Policy Store can be encrypted using a passphrase. If you wish to do this you can enter a password in this field when connecting to the Management Service interface. You must use this password thereafter when connecting to this interface.

Database

Policies and configuration information can be stored in a relational database. The SOA Security Gateway supports the following databases:

- Oracle
- SQL Server
- MySQL

Please consult the CA support team for more information on storing your configuration information in a particular database.

JDBC URL:

You must enter the JDBC URL for the database in this field.

User name:

Enter the user name to use to connect to the database in this field.

Password:

Enter the password for this user.

Passphrase:

All sensitive data (password, keys etc.) stored in the Policy Store can be encrypted using a passphrase. If you wish to do this you can enter a password in this field when connecting to the Management Service interface. You must use this password thereafter when connecting to this interface.

Directory Server

Policies and configuration information can be stored in a Director Server (LDAP directory). The SOA Security Gateway supports the following databases:

- Sun iPlanet
- Siemens DirX
- IBM Directory Server
- Microsoft Active Directory

Configure the following fields to connect to a Policy Store that is persisted in a Directory Server.

LDAP Server URL:

Enter the location of the LDAP directory server in this field. The URL is a combination of the protocol (i.e. LDAP), the IP address of the host machine, and the port number for the LDAP service. For example,

```
LDAP://IP Address:Port
```

Base DN:

You must enter the top level of the LDAP directory tree where the configuration details are stored in this field. This is normally in the format o="development", c=US.

User Name:

Enter the user name to use when accessing the LDAP directory in the **User name** field.

Password:

Enter this user's password in the **Password** field

Realm:

Enter the realm to which this user should authenticate to in the **Realm** field. This is only required in cases where the LDAP directory has multiple realms. Your directory administrator can provide the name of the appropriate realm.

SSL Enabled:

Check the **SSL Enabled** checkbox to force the SOA Security Gateway Management Console to connect to the LDAP directory over SSL.

Passphrase Key:

All sensitive data (password, keys etc.) stored in the LDAP directory server can be encrypted using a passphrase. You must enter this passphrase (if required) when connecting to the LDAP directory server.

XML Database (Tamino)

Policies and configuration information can be stored in an XML database, such as Soft-

ware AG's Tamino. Complete the following fields to connect to a Policy Store that has been stored in an XML database.

URL:

Enter the URL of the XML database in this field.

User Name:

Enter the user name to use when accessing the XML database in the **User name** field.

Password:

Enter this user's password in the **Password** field

Passphrase Key:

All sensitive data (password, keys etc.) stored in the LDAP directory server can be encrypted using a passphrase. You must enter this passphrase (if required) when connecting to the LDAP directory server.

CA Contact Details

Contact Details

For online technical assistance and a complete list of locations, primary service hours, and telephone numbers, contact Technical Support at http://ca.com/support.

Message Attribute Reference

attribute.lookup.list

Name	attribute.lookup.list
Description	User attributes can be retrieved from a variety of sources, including LDAP directories, databases, CA Entity Store, SAML attribute assertions, and so on. All retrieved attributes are stored in the attribute.lookup.list attribute, where they can be looked up at a later stage in the circuit.
Type	java.util.HashMap
Generated By	Retrieve Attributes from Database Retrieve Attributes from Directory Server Retrieve Attributes from User Store
Consumed By	
Required By	Attribute Authorization

attribute.subject.format

Name	attribute.subject.format
Description	The format of the subject ID that is used to lookup attributes, for example, X.509 DName or username.
Type	java.lang.String
Generated By	Retrieve Attributes from Directory Server
Consumed By	
Required By	

attribute.subject.id

Name	attribute.subject.id
Description	The ID of the subject that is used to look up user attributes. This can either be an X.509 DName or a username.
Type	java.lang.String
Generated By	Retrieve Attributes from Directory Server
Consumed By	
Required By	

authentication.cert

Name	authentication.cert
Description	The certificate that was used to authenticate the client,
Type	java.security.cert.X509Certificate
Generated By	SSL Authentication
Consumed By	
Required By	

authentication.issuer.format

Name	authentication.issuer.format
Description	The format of the authentication.issuer.id attribute.
Type	java.lang.String
Generated By	SSL Authentication
Consumed By	
Required By	

authentication.issuer.id

Name	authentication.issuer.id
Description	Contains the ID of the issuer of the authenticated client's certificate. This is usually either the X.509 DName or the username of the issuer of the subject's certificate.
Type	java.lang.String
Generated By	SSL Authentication
Consumed By	
Required By	

authentication.issuer.orig.format

Name	authentication.issuer.orig.format
Description	Format of the authentication.issuer.orig.id attribute. This is the format of the issuer's ID before any credential mapping was done to the identifier, e.g. DName to username.
Type	java.lang.String
Generated By	
Consumed By	
Required By	

authentication.issuer.orig.id

Name	authentication.issuer.orig.id
Description	The ID of the issuer of the subject's credential before any credential mapping took place. An example of credential mapping would involve mapping an issuer's username to a DName.
Type	java.lang.String
Generated By	

Consumed By	
Required By	

authentication.method

Name	authentication.method
Description	The method used by the client to authenticate to SOA Security Gateway.
Type	java.lang.String
Generated By	HTTP Basic Authentication SSL Authentication XML-Signature Authentication
Consumed By	
Required By	

authentication.subject.format

Name	authentication.subject.format
Description	The format of the subject's ID, e.g. X.509 DName or username.
Type	java.lang.String
Generated By	Retrieve Attributes from User Store Retrieve Attributes from Directory Server Retrieve Attributes from Database HTTP Basic Authentication SSL Authentication XML-Signature Authentication

Consumed By	
Required By	Retrieve Attributes from Database Retrieve Attributes from Directory Server Retrieve Attributes from User Store

authentication.subject.id

Name	authentication.subject.id
Description	Contains the ID of the authenticated subject.
Type	java.lang.String
Generated By	Retrieve Attributes from User Store Retrieve Attributes from Database Retrieve Attributes from Directory Server HTTP Basic Authentication SSL Authentication XML-Signature Authentication
Consumed By	
Required By	Retrieve Attributes from User Store Retrieve Attributes from Database Retrieve Attributes from Directory Server

authentication.subject.orig.format

Name	authentication.subject.orig.format
Description	The ID of the subject before any credential mapping was performed. For example, the subject's ID may be mapped from an X.509 DName to the username stored in an external database. In this case, the subject's original ID was a DName.
Type	java.lang.String
Generated By	HTTP Basic Authentication
Consumed By	
Required By	

authentication.subject.orig.id

Name	authentication.subject.orig.id
Description	Contains the ID of the authenticated subject before credential mapping was performed, e.g. from username to DName
Type	java.lang.String
Generated By	HTTP Basic Authentication
Consumed By	
Required By	

authentication.subject.password

Name	authentication.subject.password
Description	If a user authenticates to SOA Security Gateway using a username and password combination (either with HTTP digest/basic authentication or with a WS-Security Username token), the user's password is stored in this attribute.
Type	java.lang.String

Generated By	HTTP Basic Authentication
Consumed By	
Required By	

certificate

Name	certificate
Description	This message attribute can be used to store a certificate in addition to the certificate stored in the authentication.cert attribute. For example, the **Find Certificate** filter can extract a certificate from an LDAP directory, HTTP header, or the User Store. By default, it sets this certificate to the certificate attribute.
Type	java.lang.String
Generated By	Find Certificate Integrity XML-Signature Verification SSL Authentication XML-Signature Authentication
Consumed By	
Required By	

certificates

Name	certificates
Description	Contains an array of X.509 certificates for use in the **Certificate Chain Check** and **Certificate Validation** filters.
Type	java.util.ArrayList
Generated By	Find Certificate

	Integrity XML-Signature Verification SSL Authentication XML-Signature Authentication
Consumed By	
Required By	Certificate Chain File-based CRL Certificate Validation LDAP-based CRL Certificate Validation OCSP Certificate Validation XKMS Certificate Validation

content.body

Name	content.body
Description	Contains the parsed body of the incoming HTTP request.
Type	com.vordel.mime.Body
Generated By	
Consumed By	
Required By	Attachment Filtering XML Complexity Connection Content Validation Integrity XML-Signature Verification Maximum Messages

	Operation Name Resolver Reflect Reflect Message and Attributes Retrieve Attribute from HTTP Header Retrieve Attribute from Message Schema Validation Sign Message SSL Authentication Stylesheet Conversion XML-Decryption XML-Encryption XML-Signature Authentication

decryption.properties

Name	decryption.properties
Description	Specifies the XML-Encrypted block(s) to decrypt. The actual decryption is performed by the **XML-Decryption** filter.
Type	java.util.Map
Generated By	XML-Decryption Settings
Consumed By	
Required By	XML-Decryption

encryption.properties

Name	encryption.properties
Description	Allows the user to encrypt (part of) the message for a number of recipients such that only those recipients will be able to decrypt the encrypted data. The encryption is actually performed by the **XML-Encryption** filter.
Type	java.util.Map
Generated By	XML-Encryption Settings
Consumed By	
Required By	XML-Encryption

http.destination.host

Name	http.destination.host
Description	The host on which the destination Web Service is running.
Type	java.lang.String
Generated By	Dynamic Router Static Router
Consumed By	
Required By	Connection

http.destination.port

Name	http.destination.port
Description	The port on which the destination Web Service is listening.
Type	java.lang.String
Generated By	Dynamic Router Static Router
Consumed By	

Required By	Connection

http.destination.protocol

Name	http.destination.protocol
Description	Specifies the protocol to use when routing messages to the destination Web Service. Typically, the protocol is either HTTP or HTTPS.
Type	java.lang.String
Generated By	Dynamic Router Static Router
Consumed By	
Required By	Connection

http.headers

Name	http.headers
Description	This attribute contains a list of all HTTP headers from the incoming request.
Type	com.vordel.mime.HeaderSet
Generated By	Connection
Consumed By	
Required By	Add HTTP Header Connection HTTP Basic Authentication Reflect Reflect Message and Attributes Remove HTTP Header

	SOAPAction Validate HTTP Headers

http.response.info

Name	http.response.info
Description	Contains the "reason-phrase" from the HTTP status-line, e.g. "Not found", from the "404 Not found" status-line.
Type	java.lang.String
Generated By	Connection
Consumed By	
Required By	

http.response.status

Name	http.response.status
Description	Stores the HTTP status-code of the response from the Web Service, e.g. "404", from the "404 Not found" status-line.
Type	java.lang.Integer
Generated By	Connection
Consumed By	
Required By	

http.response.version

Name	http.response.version
Description	Stores the HTTP version used in the response from the Web Service.
Type	java.lang.String
Generated By	Connection

Consumed By	
Required By	

http.request.uri

Name	http.request.uri
Description	The URI on which the HTTP request was received by SOA Security Gateway.
Type	java.net.URI
Generated By	
Consumed By	
Required By	Relative Path Operation Name Connection Dynamic Router Rewrite URL

http.request.clientaddr

Name	http.request.clientaddr
Description	Contains the IP address of the client machine from which the HTTP request was sent to SOA Security Gateway.
Type	java.net.InetSocketAddress
Generated By	
Consumed By	
Required By	IP Address

http.request.verb

Name	http.request.verb
Description	Contains the HTTP verb used in the client HTTP request to SOA Security Gateway.
Type	java.lang.String
Generated By	
Consumed By	
Required By	Connection HTTP Basic Authentication

http.request.connection.error

Name	http.request.connection.error
Description	If an error occurs when SOA Security Gateway is routing on to the target Web Service, the error status will be recorded in this attribute.
Type	java.lang.String
Generated By	Connection
Consumed By	
Required By	

soap.request.method

Name	soap.request.method
Description	Stores the SOAP operation name. This is the first element under the SOAP body for an RPC encoded SOAP message.
Type	java.lang.String
Generated By	Operation Name
Consumed By	
Required By	

soap.request.method.namespace

Name	soap.request.method.namespace

Description	Contains the namespace of the element identified by the value of the soap.request.method attribute. In other words, it specifies the namespace of the SOAP operation.
Type	java.lang.String
Generated By	Operation Name
Consumed By	
Required By	

soasecuritymanager.agent.name

Name	soasecuritymanager.agent.name
Description	This attribute contains the name of the agent used by SOA Security Gateway to connect to the CA SOA Security Manager.
Type	java.lang.String
Generated By	CA SOA Security Manager Authentication
Consumed By	
Required By	CA SOA Security Manager Authorization

soasecuritymanager.action

Name	soasecuritymanager.action
Description	Contains the action to take.
Type	java.lang.String
Generated By	
Consumed By	
Required By	

soasecuritymanager.resource

Name	soasecuritymanager.resource

Description	This attribute contains the name of the resource that the client is attempting to access.
Type	java.lang.String
Generated By	
Consumed By	
Required By	

soasecuritymanager.decision

Name	soasecuritymanager.decision
Description	This attribute contains the authentication/authorization decision made by CA SOA Security Manager.
Type	java.lang.String
Generated By	CA SOA Security Manager Authentication
Consumed By	
Required By	CA SOA Security Manager Authorization

soasecuritymanager.realmdef

Name	soasecuritymanager.realmdef
Description	Contains the authentication/authorization realm of the CA SOA Security Manager.
Type	java.lang.String
Generated By	CA SOA Security Manager Authentication
Consumed By	
Required By	CA SOA Security Manager Authorization

soasecuritymanager.resource.context

Name	soasecuritymanager.resource.context
Description	This attribute describes the context of the resource that the user is attempting to ac-

	cess.
Type	java.lang.String
Generated By	CA SOA Security Manager Authentication
Consumed By	
Required By	CA SOA Security Manager Authorization

Message Filter Reference

True

Name	True
Description	Forces a true result from a policy path.
Category	Utility
License	Standard
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	True Filter

Abort

Name	Abort
Description	Forces a policy path to abort and throw an exception. This causes a SOAP Fault to be returned to the client.
Category	Utility
License	Standard
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	Abort Filter

Copy/Modify Attributes

Name	Copy/Modify Attributes
Description	This filter can be used to copy the value of one message attribute to another.
Category	Utility
License	Standard
Required Attributes	

Consumed Attributes	
Generated Attributes	
Tutorial	Copy/Modify Attributes

Trace

Name	Trace
Description	Forces SOA Security Gateway to trace the current message attributes to the configured trace destination. By default, trace files are written to the /trace directory of your SOA Security Gateway installation.
Category	Utility
License	Standard
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	Trace Message Attributes

Pause

Name	Pause
Description	This filter forces the policy to suspend processing for a specified time interval. Once this interval has elapsed, the next filter in the policy path will be executed immediately.
Category	Utility
License	Standard
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	Pause Filter

Reflect

Name	Reflect
Description	Echoes the request body back to the client.
Category	Utility
License	Standard
Required Attributes	content.body http.headers
Consumed Attributes	
Generated Attributes	
Tutorial	Reflect Message

Reflect Message and Attributes

Name	Reflect Message and Attributes
Description	Echoes the request body and the current message attributes back to the client.
Category	Utility
License	Standard
Required Attributes	content.body http.headers
Consumed Attributes	
Generated Attributes	
Tutorial	Reflect Message and Attributes

False

Name	False
Description	Forces the policy path to return false.
Category	Utility

License	Standard
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	False Filter

Attribute Authorization

Name	Attribute Authorization
Description	This filter checks the values of user attributes that are stored in the attribute.lookup.list message attribute.
Category	Authorization
License	Standard
Required Attributes	attribute.lookup.list
Consumed Attributes	
Generated Attributes	
Tutorial	Attribute Authorization

Certificate Attributes Authorization

Name	Certificate Attributes Authorization
Description	Authorizes a user by examining the attributes in that user's X.509 certificate.
Category	Authorization
License	Standard
Required Attributes	authentication.subject.id authentication.subject.format
Consumed Attributes	
Generated Attributes	
Tutorial	Certificate Attributes Authorization

Alert

Name	Alert
Description	Sends an alert to a configured alerting destination.
Category	Alert
License	Standard
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	Alert

Retrieve Attributes from Directory Server

Name	Retrieve Attribute from Directory Server
Description	Retrieves user attributes from an LDAP directory.
Category	Attributes
License	Standard
Required Attributes	authentication.subject.id authentication.subject.format
Consumed Attributes	
Generated Attributes	attribute.lookup.list attribute.subject.format attribute.subject.id
Tutorial	Retrieve Attribute from Directory Server

Retrieve Attribute from HTTP Header

Name	Retrieve Attribute from HTTP Header

Description	Retrieves the value of a HTTP header and sets it to a user-specified message attribute.
Category	Attributes
License	Standard
Required Attributes	content.body
Consumed Attributes	
Generated Attributes	
Tutorial	Retrieve Attribute from HTTP Header

Retrieve Attributes from Database

Name	Retrieve Attributes from Database
Description	Retrieves user attributes from a specified database.
Category	Attributes
License	Standard
Required Attributes	authentication.subject.id authentication.subject.format
Consumed Attributes	
Generated Attributes	attribute.lookup.list authentication.subject.id authentication.subject.format
Tutorial	Retrieve Attributes from Database

Retrieve Attribute from Message

Name	Retrieve Attribute from Message

Description	Retrieves the value of an XML attribute or element from the message and sets it to a user-specified message attribute.
Category	Attributes
License	Standard
Required Attributes	content.body
Consumed Attributes	
Generated Attributes	
Tutorial	Retrieve Attribute from Message

Retrieve Attributes from User Store

Name	Retrieve from User Store
Description	Retrieves user attributes from the User Store and stores them in the attribute.lookup.list message attribute.
Category	Attributes
License	Standard
Required Attributes	authentication.subject.id
Consumed Attributes	
Generated Attributes	attribute.lookup.list authentication.subject.id authentication.subject.format
Tutorial	Retrieve Attributes from User Store

Operation Name

Name	Operation Name
Description	Compares the first element of the SOAP body (i.e. the SOAP operation) and its

	namespace to the values configured here.
Category	Resolvers
License	Standard
Required Attributes	content.body
Consumed Attributes	
Generated Attributes	soap.request.method soap.request.method.namespace
Tutorial	Operation Name

Relative Path

Name	Relative Path
Description	Matches the relative path (i.e. uri) on which the request was received to the value configured here.
Category	Resolvers
License	Standard
Required Attributes	http.request.uri
Consumed Attributes	
Generated Attributes	
Tutorial	Relative Path

SOAPAction

Name	SOAPAction
Description	Matches the SOAPAction HTTP header on the incoming request to the value configured in this filter.
Category	Resolvers
License	Standard
Required Attributes	http.headers

Consumed Attributes	
Generated Attributes	
Tutorial	SOAP Action

Sign Message

Name	Sign Message
Description	Signs the selected part of the incoming request.
Category	Integrity
License	Standard
Required Attributes	content.body
Consumed Attributes	
Generated Attributes	
Tutorial	Sign Message

Connection

Name	Connection
Description	This filter is responsible for connecting to the target Web Service or system. SOA Security Gateway can mutually authenticate to the endpoint using SSL certificates or HTTP basic/digest authentication.
Category	Routing
License	Standard
Required Attributes	content.body http.destination.host http.destination.port http.destination.protocol

	http.headers http.request.uri http.request.verb
Consumed Attributes	
Generated Attributes	http.headers http.request.connection.error http.response.info http.response.status http.response.version
Tutorial	Connection

Dynamic Router

Name	Dynamic Router
Description	In cases where SOA Security Gateway is acting as a proxy, it can extract the URL from the request line of the HTTP request and route the message to this address.
Category	Routing
License	Standard
Required Attributes	http.request.uri
Consumed Attributes	
Generated Attributes	http.destination.host http.destination.port http.destination.protocol

Tutorial	Dynamic Router

Rewrite URL

Name	Rewrite URL
Description	In cases where SOA Security Gateway is acting as a proxy, it can forward the message on to the address specified in the request line of the HTTP request. This filter can be used to rewrite the URL of the original request to an alternative one, i.e. service virtualization.
Category	Routing
License	Standard
Required Attributes	http.request.uri
Consumed Attributes	
Generated Attributes	
Tutorial	Rewrite URL

Static Router

Name	Static Router
Description	The static router is used to configure connection details for a particular endpoint. SOA Security Gateway will route messages to the endpoint configured here.
Category	Routing
License	Standard
Required Attributes	
Consumed Attributes	
Generated Attributes	http.destination.host http.destination.port

	http.destination.protocol
Tutorial	Static Router

Log Access

Name	Log Access
Description	Logs message details in Common Log Format to an Access Log. The log file is written to the /logs directory of your SOA Security Gateway installation.
Category	Log
License	Standard
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	Log Access

Log Message Payload

Name	Log Message Payload
Description	Logs the message payload, including HTTP headers and MIME/DIME attachments, at a particular point in the policy.
Category	Log
License	Standard
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	Log Message Payload

SOAP Fault

Name	SOAP Fault

Description	If a **SOAP Fault** handler is configured for a policy, it handles any exceptions that occur in the policy. As such it dictates the format of the SOAP Fault that is returned to the client.
Category	Fault Handlers
License	Standard
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	SOAP Fault

XML-Decryption

Name	XML-Decryption
Description	Decrypts an XML-Encrypted message according to the settings configured in the **XML-Decryption Settings** filter. These settings are stored in the decryption.properties message attribute, which is a required attribute for this filter.
Category	Encryption
License	Standard
Required Attributes	content.body decryption.properties
Consumed Attributes	
Generated Attributes	
Tutorial	XML-Decryption

XML-Decryption Settings

Name	XML-Decryption Settings
Description	This filter is used to specify the XML-Encrypted blocks to decrypt in the mes-

	sage. All of the encrypted blocks can be decrypted or a single encrypted block can be selected using an XPath expression. The actual decryption is performed by the **XML-Decryption** filter.
Category	Encryption
License	Standard
Required Attributes	
Consumed Attributes	
Generated Attributes	decryption.properties
Tutorial	XML-Decryption Settings

XML-Encryption

Name	XML-Encryption
Description	Encrypts (part of) an XML message as specified in the **XML Encryption Settings** filter. The message will be encrypted such that only its intended recipients can decrypt it.
Category	Encryption
License	Standard
Required Attributes	content.body encryption.properties
Consumed Attributes	
Generated Attributes	
Tutorial	XML-Encryption

XML-Encryption Settings

Name	XML-Encryption Settings
Description	This filter is used to specify the part(s) of

	the message to encrypt, and for whom the message is to be encrypted. Only the intended recipients will be able to decrypt the message.
Category	Encryption
License	Standard
Required Attributes	
Consumed Attributes	
Generated Attributes	encryption.properties
Tutorial	XML-Encryption Settings

Add HTTP Header

Name	Add HTTP Header
Description	Adds a user-specified HTTP header to the downstream message.
Category	Conversion
License	Standard
Required Attributes	http.headers
Consumed Attributes	
Generated Attributes	
Tutorial	Add HTTP Header

Remove HTTP Header

Name	Remove HTTP Header
Description	Removes a user-specified HTTP header from the downstream message.
Category	Conversion
License	Standard
Required Attributes	http.headers

Consumed Attributes	
Generated Attributes	
Tutorial	Remove HTTP Header

Set Message

Name	Set Message
Description	Sets the content of the message payload.
Category	Conversion
License	Standard
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	Set Message

Stylesheet Conversion

Name	Stylesheet Conversion
Description	This filter uses an XSLT stylesheet to convert the body of the incoming request to an alternative XML grammar or format..
Category	Conversion
License	Standard
Required Attributes	> content.body
Consumed Attributes	
Generated Attributes	
Tutorial	Stylesheet Conversion

Policy Shortcut

Name	Policy Shortcut
Description	This filter can be used to pass control to

	another policy. It is very useful for creating a **policy macro** that contains small pieces of logic that you may wish to keep outside of a policy so that it can be re-used. This helps to keep the main logic of a policy un-cluttered.
Category	Utility
License	Standard
Required Attributes	The required attributes for this filter are whatever attributed are required by the **start node** of the policy shortcut.
Consumed Attributes	
Generated Attributes	The generated attributes for this filter are the attributes that are returned from the **end node** of the policy shortcut.
Tutorial	Policy Shortcut

Attachment Filtering

Name	Attachment Filtering
Description	Filters MIME and DIME messages based on the types of their attachments.
Category	Content Filtering
License	Standard
Required Attributes	content.body
Consumed Attributes	
Generated Attributes	
Tutorial	Content Type Filter

XML Complexity

Name	XML Complexity
Description	Checks the depth and complexity of XML messages.
Category	Content Filtering

License	Standard
Required Attributes	content.body
Consumed Attributes	
Generated Attributes	
Tutorial	XML Complexity

Content Validation

Name	Content Validation
Description	Runs a boolean XPath expression on the incoming request.
Category	Content Filtering
License	Standard
Required Attributes	content.body
Consumed Attributes	
Generated Attributes	
Tutorial	Content Validation

Integrity XML-Signature Verification

Name	Integrity XML-Signature Verification
Description	Verifies the integrity of the incoming message by validating its XML-Signature. This ensures that the message was not tampered with after it was signed.
Category	Content Filtering
License	Standard
Required Attributes	content.body
Consumed Attributes	
Generated Attributes	certificate

	certificates
Tutorial	Integrity XML-Signature Verification

Maximum Messages

Name	Maximum Messages
Description	Limits the number of messages a client can send in a specified interval through the policy in which this filter is configured. In other words, it provides filtering of messages on a per client, per service basis.
Category	Content Filtering
License	Standard
Required Attributes	content.body
Consumed Attributes	
Generated Attributes	
Tutorial	Maximum Messages

Schema Validation

Name	Schema Validation
Description	Validates the contents of the message body against a selected XML Schema. This ensures that the message adheres to the correct message format, and can also ensure that the message contains appropriate data.
Category	Content Filtering
License	Standard
Required Attributes	content.body
Consumed Attributes	
Generated Attributes	

Tutorial	Schema Validation

Validate HTTP Headers

Name	Validate HTTP Headers
Description	Filters MIME and DIME messages based on the types of their attachments.
Category	Content Filtering
License	Standard
Required Attributes	http.headers
Consumed Attributes	
Generated Attributes	
Tutorial	Validate HTTP Headers

Validate Message Attribute

Name	Validate Message Attribute
Description	Compares the value of a message attribute to a configured regular expression. It can also check for the presence of **threatening content** regular expressions such as SQL injection and buffer overflow attacks.
Category	Content Filtering
License	Standard
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	Validate Message Attributes

Certificate Revocation List (File)

Name	Certificate Revocation List (File)
Description	Looks up a user's certificate in a file-based

	CRL to see if that user has been revoked.
Category	Certificate
License	Standard
Required Attributes	certificates
Consumed Attributes	
Generated Attributes	
Tutorial	Certificate Revocation List (File)

Certificate Revocation List (LDAP)

Name	Certificate Revocation List (LDAP)
Description	Looks up a user's certificate in an LDAP-based CRL to see if that user has been revoked.
Category	Certificate
License	Standard
Required Attributes	certificates
Consumed Attributes	
Generated Attributes	
Tutorial	Certificate Revocation List (LDAP)

Certificate Chain

Name	Certificate Chain
Description	Ensures that a trusted CA (Certificate Authority) issued the certificate. Trusted CA certificates are stored in the CA Trusted Certificate Store.
Category	Certificate
License	Standard
Required Attributes	certificates

Consumed Attributes	
Generated Attributes	
Tutorial	Certificate Chain

Find Certificate

Name	Find Certificate
Description	Locates a certificate from a message attribute, HTTP header, message attachment, or extracts a certificate from the User Store. The extracted certificate is stored in a user-specified message attribute. This new attribute will then appear as a **Generated Attribute** in the policy. The certificate is stored in the certificate attribute by default.
Category	Certificate
License	Standard
Required Attributes	
Consumed Attributes	
Generated Attributes	certificate certificates
Tutorial	Find Certificate

OCSP (Online Certificate Status Protocol)

Name	OCSP Certificate Validation
Description	Checks the status of a user's certificate against a group of OCSP responders.
Category	Certificate
License	Standard
Required Attributes	certificates

Consumed Attributes	
Generated Attributes	
Tutorial	OCSP

XKMS (XML Key Management and Security)

Name	XKMS Certificate Validation
Description	Validates a user's certificate against a group of XKMS responders.
Category	Certificates
License	Standard
Required Attributes	certificates
Consumed Attributes	
Generated Attributes	
Tutorial	XKMS

HTTP Basic Authentication

Name	HTTP Basic Authentication
Description	Authenticates a client against a configured user store using HTTP basic authentication.
Category	Authentication
License	Standard
Required Attributes	http.headers http.request.verb
Consumed Attributes	
Generated Attributes	authentication.method authentication.subject.format authentication.subject.id

	authentication.subject.orig.format authentication.subject.orig.id authentication.subject.password
Tutorial	HTTP Basic Authentication

IP Address

Name	IP Address
Description	Allows or denies access to an IP address or range of IP addresses.
Category	Authentication
License	Standard
Required Attributes	http.request.clientaddr
Consumed Attributes	
Generated Attributes	
Tutorial	IP Address

XML-Signature Authentication

Name	XML-Signature Authentication
Description	SOA Security Gateway can authenticate a client by validating the XML-Signature on an incoming request. A successful signature validation proves that the client had access to the private key that was used to sign the request.
Category	Authentication
License	Standard
Required Attributes	content.body
Consumed Attributes	

Generated Attributes	authentication.cert authentication.issuer.format authentication.issuer.id authentication.method authentication.subject.format authentication.subject.id certificate certificates
Tutorial	XML-Signature Authentication

SSL Authentication

Name	SSL Authentication
Description	Authenticates a user's SSL certificate.
Category	Authentication
License	Standard
Required Attributes	content.body
Consumed Attributes	
Generated Attributes	authentication.cert authentication.issuer.format authentication.issuer.id authentication.method authentication.subject.format

	authentication.subject.id certificate certificates
Tutorial	SSL Authentication

Insert WS-Addressing

Name	Insert WS-Addressing
Description	Inserts WS-Addressing information into a SOAP message.
Category	Connection
License	Standard
Required Attributes	http.destination.host http.destination.port http.destination.protocol http.headers http.request.uri soap.action
Consumed Attributes	
Generated Attributes	
Tutorial	SSL Authentication

Read WS-Addressing

Name	Read WS-Addressing
Description	Uses WS-Addressing information contained

	within a SOAP message to route the message.
Category	Connection
License	Standard
Required Attributes	content.body
Consumed Attributes	
Generated Attributes	http.destination.host http.destination.port http.destination.protocol http.request.uri
Tutorial	SSL Authentication

Set Response Status

Name	Set Response Status
Description	Explicitly sets the response status of a given message.
Category	Utility
License	Standard
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	Set Response Status

CA SOA Security Manager Authorization

Name	CA SOA Security Manager Authorization
Description	Authorizes an authenticated user against CA SOA Security Manager.

Category	Authorization
License	Connector
Required Attributes	http.request.clientaddr soasecuritymanager.agent.name soasecuritymanager.decision soasecuritymanager.realmdef soasecuritymanager.resource.context
Consumed Attributes	
Generated Attributes	attribute.lookup.list attribute.subject.format attribute.subject.id
Tutorial	CA SOA Security Manager Authorization

CA SOA Security Manager Authentication

Name	CA SOA Security Manager Authentication
Description	Authenticates a user against CA SOA Security Manager.
Category	Authentication
License	Standard
Required Attributes	content.body http.request.clientaddr http.request.uri http.request.verb

Consumed Attributes	
Generated Attributes	authentication.method authentication.subject.format authentication.subject.id soasecuritymanager.agent.name soasecuritymanager.decision soasecuritymanager.realmdef soasecuritymanager.resource.context
Tutorial	CA SOA Security Manager Authentication

Glossary of Terms

XPath

XML Path Language (XPath), is a language that describes how to locate and process specific parts of an XML document. See the XML Path Language Specification [http://www.w3.org/TR/xpath] for more details.

SSL

Secure Sockets Layer (SSL) is an encrypted communications protocol for sending information securely across the Internet. It sits just above the transport layer, and below the application layer and transparently handles the encryption and decryption of data when a client establishes a secure connection to the server. It optionally provides peer entity authentication between client and server.

XSL

XML Stylesheet Language is used to convert XML documents into different formats, the most common of which is HTML. In a typical scenario, an XML document will reference an XSL stylesheet, which will define how the XML elements of the document should be displayed as HTML. Therefore, a clear separation of content and presentation is achieved.

PKCS#12

PKCS#12 is a standard for storing private keys and certificates securely. It is used in (among other things) Netscape and Microsoft Internet Explorer with their import and export options.

keystore

In JDK 1.2/1.4, the keystore file contains your public and private keys. It has a file name ".keystore", the peculiar leading dot makes the file read-only in Unix. It is stored in PKCS #12 format containing both public and private keys, protected by a passphrase.

cacert

A file used to keep the root certificates of signing authorities. The default password is *changeit*. It is typically stored in `c:\jdk1.4\jre\lib\security\cacerts`. Each entry is identified by a unique alias, and is either a key entry or a certificate entry. Key entries consist of a key pair, whereas certificate entries consist of just a certificate.

Since you implicitly trust all the Certificate Authorities in the cacerts file for code signing and verification, you must manage the cacerts file carefully. The cacerts file should contain only certificates of the CAs you trust.

WSDL

Web Services Description Language (WSDL) is an XML format for describing network services as a set of endpoints operating on messages containing either docu-

ment-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services).

WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME. See the Web Services Description Language Specification [http://www.w3.org/TR/wsdl] for more details.

UDDI

Universal Description, Discovery, and Integration (UDDI) is an XML-based lookup service for locating Web Services in an Internet scenario. See the Universal Description Discovery Integration (UDDI) standard [http://www.uddi.org/] for more details.

DOM

Document Object Model (DOM) is a generic interface (platform- and language-neutral) that allows external programs to edit a document's contents, structure, and style.

Signature

A value computed with a cryptographic algorithm and appended to a data object in such a way that any recipient of the data can use the signature to verify the data's origin and integrity.

Base64

A method of encoding 8-bit characters as ASCII printable characters. It is typically used to encode binary data so that it may be sent over text-based protocols such as HTTP and SMTP. Base64 is a scheme where 3 bytes are concatenated, then split to form 4 groups of 6-bits each; and each 6-bits gets translated to an encoded printable ASCII character, via a table lookup. The specification is described in RFC 2045.

X509

X509 is the standard which defines the contents and data format of a public-key certificate.

OCSP

Online Certificate Status Protocol is an automated certificate checking network protocol. A client will query the OCSP responder for the status of a certificate. The responder returns whether the certificate is still trusted by the CA that issued it.

CRL

A Certificate Revocation List is a signed list indicating a set of certificates that are no longer considered valid by the certificate issuer. CRLs may be used to identify revoked public-key certificates or attribute certificates and may represent revocation of certificates issued to authorities or to users. The term CRL is also commonly used as a generic term applying to different types of revocation lists.

XKMS

XML Key Management Specification (XKMS) uses the relative simplicity of XML to provide key management services so that a Web Service can query the trustworthiness of a user's certificate over the Internet. XKMS aims to simplify application building by separating digital-signature handling and encryption from the applications themselves. See the XML Key Management Specification [http://www.w3.org/TR/xkms/] for more details.

Public-Key

The publicly-disclosable component of a pair of cryptographic keys used for asymmetric cryptography.

Private-Key

The secret component of a pair of cryptographic keys used for asymmetric cryptography.

CA

A Certificate Authority (CA) issues digital certificates (especially X.509 certificates) and vouches for the binding between the data items in a certificate.

DName

An identifier that uniquely represents an object in the X.500 Directory Information Tree (DIT). A DName is a set of attribute values that identify the path leading from the base of the DIT to the object that is named. An X.509 public-key certificate or CRL contains a DName that identifies its issuer, and an X.509 attribute certificate contains a DN or other form of name that identifies its subject.

URI

Uniform Resource Identifiers (URIs), are a platform-independent way to specify a file or resource somewhere on the web. Strictly speaking, every URL is also a URI, but not every URI is also a URL. Two RFCs specify the format of a URI:

- RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax [http://www.faqs.org/rfcs/rfc2396.html]
- RFC 2732: Format for Literal IPv6 Addresses in URIs [http://www.faqs.org/rfcs/rfc2732.html]

SOAP

SOAP, or Simple Object Access Protocol is an XML-based object invocation protocol. SOAP was originally developed for distributed applications to communicate over HTTP and through corporate firewalls. SOAP defines the use of XML and HTTP to access services, objects and servers in a platform-independent manner. SOAP provides a way to access services, objects, and servers in a completely platform-independent manner. SOAP is a wire protocol that can be used to facilitate highly ultra-distributed architecture.

SOAP is simple. It is nothing more and nothing less than a protocol that defines how to access services, objects, and servers in a platform-independent manner using HTTP (also SMTP) and XML. See the Simple Object Access Protocol Specification [http://www.w3.org/TR/SOAP/] for more details.

ASN.1

ASN.1 (Abstract Syntax Notation One) is a standard format for transmitting messages over a network. The standard defines a syntax for describing the structure of a message, and also rules for encoding the various data types contained within the message. ASN.1 is defined in two ISO standards:

1. ISO 8824/ITU X.208 specifies the ASN.1 syntax.
2. ISO 8825/ITU X.209 specifies the encoding rules for ASN.1

ISO

ISO is a worldwide consortium of national standards bodies from more than 140 countries. The goal of ISO is to promote standardization in the world with a view to facilitating the international exchange of goods and services, and to develop cooperation in scientific, technological and economic activity.

DTD

A Document Type Definition defines a formal grammar for specifying the structure of an XML document. An XML document is said to be *valid* if it conforms to the syntactic rules specified in the DTD.

SAX

The Simple API for XML is an interface that allows applications to read in XML data. SAX is an *event-based* interface, which means that it responds to certain *events*. SAX is a read-only interface, which means that it cannot be used to generate XML elements in the same way that the DOM can. However, it is generally more efficient than DOM for reading XML documents since it does not keep the entire XML tree in memory like DOM parsing does.

XSLT

Extensible Stylesheet Language Transformations are used to convert XML documents into other formats.

LDAP

LDAP is a "lightweight" version of Directory Access Protocol (DAP), which is part of X.500, a standard for directory services in a network. An LDAP directory stores information on resources in a hierarchical fashion. This makes data retrieval very efficient.

SAML

Security Assertion Markup Language (SAML) is an XML standard for establishing

trust between entities. SAML assertions can contain identity information about users (authentication assertions) and also information about the access permissions of users (authorization assertions). The basic idea is that when a user is authenticated at one site, that site issues a SAML authentication assertion and gives it to the user. The user can then use this assertion in requests at other affiliated sites. These sites need only check the details contained within the authentication assertion in order to authenticate the user. In this way, SAML allows authentication and authorization information to be shared between separate sites.

DER

Distinguished Encoding Rules is a type of ASN.1 encoding and is widely used to define the format of X.509 certificates.

PEM

PEM (Privacy Enhanced Mail) was originally intended for securing Internet mail through authentication, message integrity and confidentiality using various encryption techniques. Its scope was widened in later years for use in a broader range of applications, such as Web Servers. Its format is essentially a base64-encoded certificate wrapped in *BEGIN CERTIFCATE* and *END CERTIFICATE* directives.

TLS

Transport Layer Security is the successor to SSL 3.0. Like SSL, it allows applications to communicate over a secure channel.