API Gateway

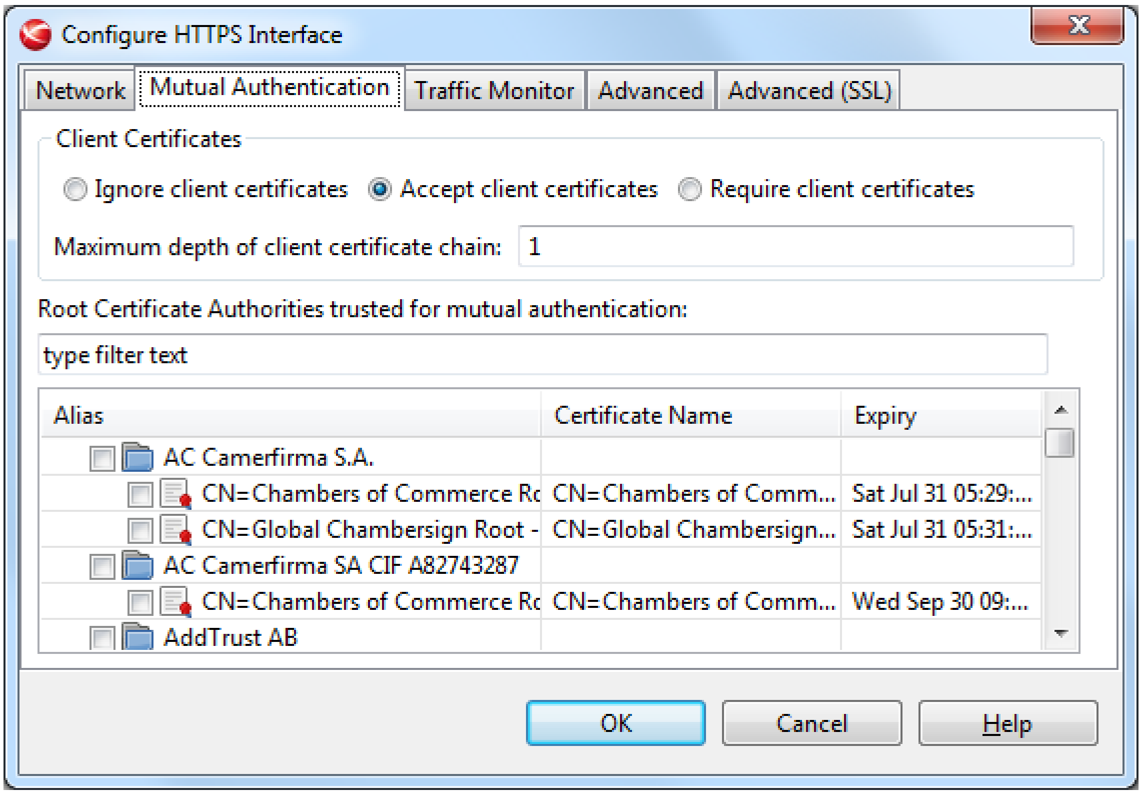
Configuring & Testing SSL Mutual Authentication

1. Introduction

This guide describes the configuration of SSL Mutual authentication for the API Gateway. While version 7.4 is depicted in the screenshots, the information is generally applicable to other API Gateway versions. Where applicable, this guide will try to call differences between versions.

2. Server Configuration

Any HTTPS port on an API Gateway can enable mutual authentication by choosing either to accept, or require client certificates on the screen picture below. This will cause the API Gateway to send a `CertificateRequest` message during the handshake and either accept or require a certificate.



Client Certificates

Ignore client certificates [✓ Accept client certificates] Require client certificates

Maximum depth of client certificate chain: 1

Root Certificate Authorities trusted for mutual authentication:

type filter text

Alias	Certificate Name	Expiry
AC Camerfirma S.A.		
CN=Chambers of Commerce Rc	CN=Chambers of Comm...	Sat Jul 31 05:29:...
CN=Global Chambersign Root -	CN=Global Chambersign...	Sat Jul 31 05:31:...
AC Camerfirma SA CIF A82743287		
CN=Chambers of Commerce Rc	CN=Chambers of Comm...	Wed Sep 30 09:...
AddTrust AB		

You then check whichever intermediate or root CAs you trust to issue client certificates. Note that every certificate selected here will be sent to every client, prior to authentication, in each and every handshake.

If you choose to trust a public CA, note that anyone who can obtain a certificate from that CA will be able to authenticate. For this reason, you may wish to issue client certificates via a CA you control.

Now imagine a certificate chain like the following:

```
[Root CA] – [Intermediate CA A] – [Intermediate CA B] – [client]
```

Here, Root is the issuer of A, A is the issuer of B, and B is the issuer of client. Root, A, and B are all CA certificates, while client is an end entity certificate.

If [Intermediate CA B] is selected above, the client only has to submit [client] during every handshake. If only [Root CA] were selected, the client would have to submit the chain Intermediate CA A] – [Intermediate CA B] – [client] to authenticate. For this reason, to avoid passing unnecessary certificates around, the gateway should be configured such that its store contains [Root CA] – [Intermediate CA A] – [Intermediate CA B] and only [Intermediate CA B] should be clicked on the tab above.

This is also related to the "maximum depth of client certificate chain" option. If [Root CA] were trusted, the client here would have to submit a 3-certificate long chain. Because that exceeds the maximum depth of 1 shown above, the connection would be rejected.

The chain must also be complete. In our example, if the API Gateway did not contain a copy of [Intermediate CA A] and was otherwise correctly configured to trust [Intermediate CA B] as the issuer, the client would only send [client]. So the connection would fail because [client] could not be chained up to a self-signed root CA (a CA certificate where the subject is identical to the issuer).

The client certificate presented can be examined by API Gateway policies. The SSL filter copies the client certificate (if any) to a message whiteboard attribute. It can then be passed to the Certificate Attributes filter to copy all of the certificate's information to whiteboard entries, or used with the various CRL filters, compared with a certificate associated with that user obtained from LDAP, etc. For more details, please refer to the User's Guide or Policy Developer's Guide for your version.

The server itself must always send a complete chain to the client, terminated by a self-signed root CA. The API Gateway will do this automatically, provided that all of the certs required exist within its certificate store. Starting from the server cert's issuer, you should be able to use the search feature and locate a certificate with a CN that is completely identical within the store. If the issuer of that certificate is not identical to its subject, you should be able to search for its issuer, as well, until you find a certificate where the issuer is identical to the subject. If any certificates are missing from this chain, you must obtain them from the CA that issued your certificate. Public CAs often offer downloads of these on their website, but you must ensure that you get the right certs. Do not import duplicate certs into the API Gateway. Having multiple copies of the same certificate can trigger known bugs in earlier copies. If the certificates use the authority & subject key identifier extensions, those must also chain correctly. That is to say, the authority key identifier of the server cert must match the subject key identifier of the issuer's certificate.

Finally, both server and client certificates must be suitable for that purpose. That can be tested within openssl, via `openssl verify –purpose [purpose] –CAfile [file containing cert chain up to issuer of the cert being tested] <your_cert>`

Where [purpose] should be either sslclient or sslserver as appropriate. Otherwise valid certificates which do not contain the proper extensions allowing them to be client certificates will not be valid.

While this is not known to differ among current OpenSSL versions, please refer to the next section to ensure that you test this with a version of OpenSSL which matches your gateway. In general, you need the appropriate extendedKeyUsage flags signifying your usage (client or server), as well as some basic key usage bits, such as 'key agreement' and 'data encipherment'. If you use legacy Netscape extensions, those also must be set correctly, but they are not required to be present.

3. OpenSSL Version Differences

Versions of API Gateway prior to 7.3 contain OpenSSL 0.9.8. API Gateway 7.3 & 7.4 contain OpenSSL 1.0.1. For that reason, only 7.3 and 7.4 support TLS 1.1 and TLS 1.2, as OpenSSL did not support those protocols until OpenSSL 1.0.1.

This also affects cipher specs used. For example, it is common to disable SSLv2 and SSLv3 via a cipher spec containing something like "`DEFAULT:!SSLv2:!SSLv3`". Although OpenSSL 0.9.8 supports TLS 1.0, all of the ciphers used in TLS 1.0 are shared with those used in SSLv3, so that cipher spec would disable all usable ciphers and cause an error in older versions of API Gateway. In API Gateway 7.3 and 7.4, it would simply effectively mandate TLS 1.2, as all remaining ciphers would be TLS 1.2 ciphers.

Note also that the API Gateway uses an embedded version of OpenSSL and ignores any OpenSSL that may be present on the system. Thus, if you run 'openssl' on the command line, it may not correspond with the actual version being used by the API Gateway. On Linux, you can invoke our embedded OpenSSL via `/apigateway/posix/bin/vrun openssl <arguments>`

To check the openssl version use: `vrun openssl version`

To check the ciphers enabled by a cipher string, run: `vrun openssl ciphers –v "<cipher string>"`

Be sure to get proper security advice regarding appropriate cipher strings. There are several traps one can fall into, for example, believing that HIGH means "high security" when it currently only has to do with key sizes, and leaves several insecure ciphers enabled which do not include authentication. Methods that do not offer authentication must be disabled via "`!aNULL`" while methods that do not offer encryption must be disabled via "`!eNULL`". Please consult the OpenSSL documentation listed in the references section for further details.

4. Testing Mutual SSL Configurations

Mutual authentication can be tested relatively easily with OpenSSL s_client. This guide will give only show a few basic commands. More documentation for OpenSSL s_client is available online and the options can be obtained via openssl s_client –help

First, to see the certificates returned by an SSL server, use: `openssl s_client –connect EXAMPLE.COM:443 –showcerts –CAfile <file containing trusted CA certs>`

This example connects to example.com on port 443 (the default HTTPS port) and will show the certificate chain returned by the server. This chain is required to follow a particular order: it starts with the server certificate, followed by the server cert's issuer, all the way up to the root certificate. Sending the root itself is optional, according to the RFCs, as the client is expected to have its own copy of the roots.

Note that you can copy paste the BEGIN/END CERTIFICATE blocks of PEM (text) encoded certificates into a text file, rename it to .p7b, and have a valid CA file.

To change that command to submit a client certificate, you do:

```
openssl s_client –connect EXAMPLE.COM:443 –CAfile <file> -cert <cert file> -key <keyfile>
```

OpenSSL will either connect properly, in which case you will be able to type in text to send to your server, or it will fail with an error message you can look up in the OpenSSL documentation available online.

5. References

OpenSSL Documentation
https://www.openssl.org/docs/

RFC 6101 - The Secure Sockets Layer (SSL) Protocol Version 3.0
http://tools.ietf.org/html/rfc6101

RFC 2246 - The TLS Protocol Version 1.0
http://tools.ietf.org/html/rfc2246

RFC 4346 - The Transport Layer Security (TLS) Protocol Version 1.1
http://tools.ietf.org/html/rfc4346

RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2
http://tools.ietf.org/html/rfc5246