How to configure HTTPS proxying in PNS 2

February 29, 2024

Abstract
This tutorial describes how to configure PNS to proxy HTTPS traffic

Table of Contents

1. Preface

This tutorial provides guidelines for PNS administrators on how to enable proxying HTTP traffic embedded into secure TLS connections. Knowledge in TCP/IP and PNS administration is required to fully comprehend the contents of this paper. The procedures and concepts described here are applicable to version 2 of PNS. Detailed information is provided to configure PNS both from Management Console and using Python scripts.

Note that explaining the concepts of the different aspects of TLS proxying is beyond the scope of this tutorial. For background information, see the following documents:

- For details on deriving and modifying proxies, see *Section 6.6, Proxy classes* in *Proxedo Network Security Suite 2 Administrator Guide*.

- For details on configuring Application-level Gateway proxies to handle TLS connections, see *How to configure TLS proxying in PNS 2*.

- For an overview on certificates and related topics in connection with PNS, see *Chapter 11, Key and certificate management in PNS* in *Proxedo Network Security Suite 2 Administrator Guide*.

- For details on the available attributes of the Application-level Gateway HTTP proxy that you can adjust and modify to best suit your needs, see *Section 4.6, Module Http* in *Proxedo Network Security Suite 2 Reference Guide*

You can download the above documents at the *Documentation Page*.

1.1. Configuring Application-level Gateway: MC versus Python

Application-level Gateway can be fully configured using either the graphical Management Console (MC) or manually by editing plain text Python configuration files. The suggestions and the background information provided in this tutorial are equally applicable to both methods. Step-by-step explanation with screenshots are given for MC-based configuration, while sample Python code lines can be found at the end of each step. After replacing the sample parameters (for example, IP addresses) with the proper ones for the actual configuration, add these lines to the policy file of Application-level Gateway (usually found under `/etc/vela/policy.py`). Also pay attention to the proper indentation of Python code lines. For more details, see *Chapter 10, Local firewall administration* in *Proxedo Network Security Suite 2 Administrator Guide*.

If Management Console is used and the Python code generated by MC needs to be displayed, select a host, then select **Configuration > View** from the main menu.

2. Configuring HTTPS proxying

For proxying HTTPS connections, a properly configured HTTP proxy is required. The HTTP proxy will handle the external TLS connection and analyze the HTTP traffic embedded into the encrypted TLS channel. Two basic scenarios — a transparent and a non-transparent one — will be discussed.

The best way is to derive own proxy classes from the default ones and modify their parameters as required. For details on deriving and modifying proxies, see _Section 6.6, Proxy classes_ in _Proxedo Network Security Suite 2 Administrator Guide_.

2.1. Procedure – Enabling TLS-encryption in the connection

Purpose:

To proxy HTTPS connections, configure an Encryption Policy to handle TLS connections, and use this Encryption Policy in your Service. The policy will be configured to:

- Require the client and the server to use strong encryption algorithms, the use of weak algorithms will not be permitted.

- Enable connections only to servers with certificates signed by CAs that are in the trusted CAs list of the PNS firewall node. (For details on managing trusted CA groups, see _Section 11.3.7.3, Managing trusted groups_ in _Proxedo Network Security Suite 2 Administrator Guide_.)

- The clients will only see the certificate of PNS. To allow the clients to access the certificate information of the server, see _Procedure 2.2, Configuring keybridging_ in _How to configure TLS proxying in PNS 2_.

Steps:

Step 1.   Generate a certificate for your firewall. The Application-level Gateway component requires its own certificate and keypair to perform TLS proxying.
MC: Create a certificate, set the firewall as the owner host of the certificate, then distribute it to the firewall host. For details, see _Chapter 11, Key and certificate management in PNS_ in _Proxedo Network Security Suite 2 Administrator Guide_.

Python: In configurations managed manually from python, create an X.509 certificate (with its related keypair) using a suitable software and deploy it to the PNS firewall host (for example, copy it to the `/etc/key.d/mycert` folder).

Step 2.   Create and configure an Encryption Policy. Complete the following steps.

Step a. Navigate to the **Application-level Gateway** MC component of the firewall host.

Step b. Select **Policies > New**.

Step c. Enter a name into the **Policy name** field, for example, `MyTLSEncryption`.
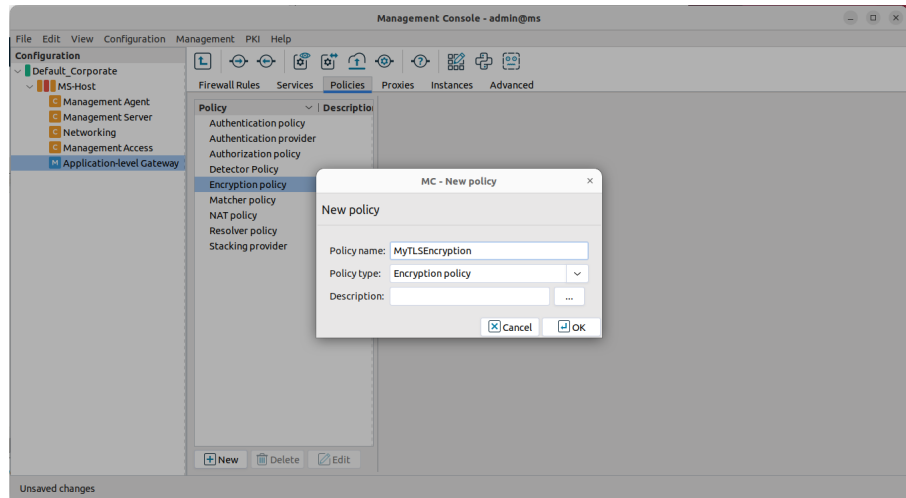
Figure 1. Creating a new Encryption policy

Step d. Select **Policy type > Encryption Policy**, then click OK.

Step e. Select **Class > TwoSidedEncryption**.
Python:

```
EncryptionPolicy(
    name="MyTLSEncryption",
    encryption=TwoSidedEncryption()
    )
```

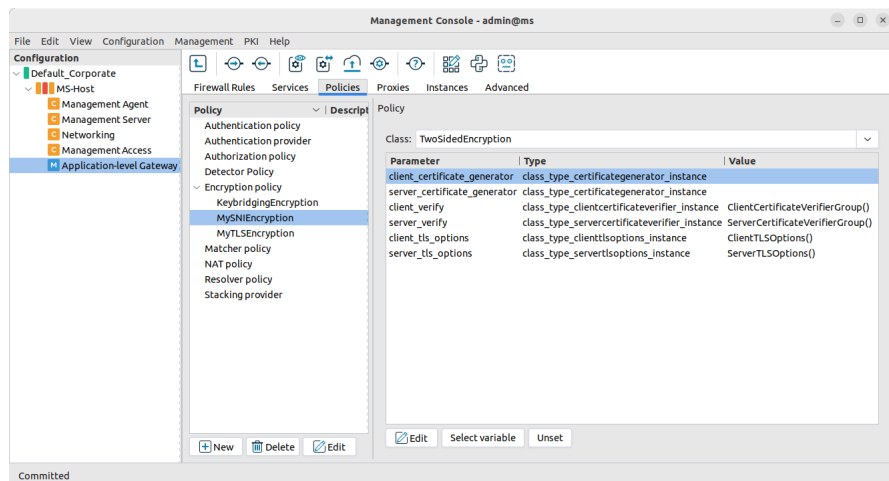Step f. Double-click **client_certificate_generator**, then select **Class > StaticCertificate**.

Figure 2. Selecting Encryption policy class

Step g. Double-click the **certificates** and click **New** to add a certificate entry to a list of certificates.
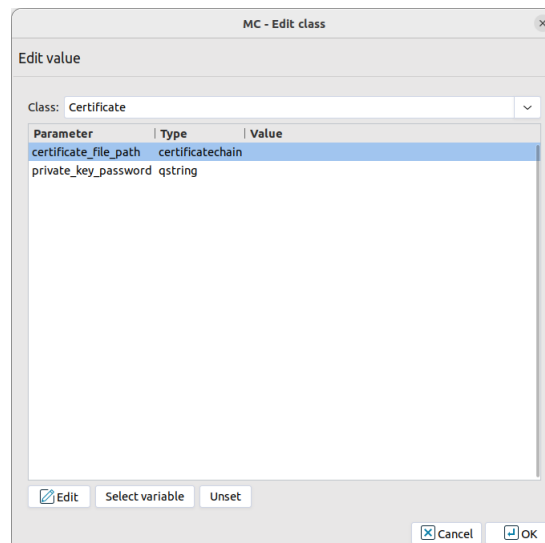
Figure 3. Creating a new certificate entry

Step h. Double-click the **certificate_file_path**. A window displaying the certificates owned by the host will open up. The lower section of the window shows the information contained in the certificate. Select the list of certificates Application-level Gateway is required to show to the clients (for example, the certificate created in Step 1), then click **Select**.
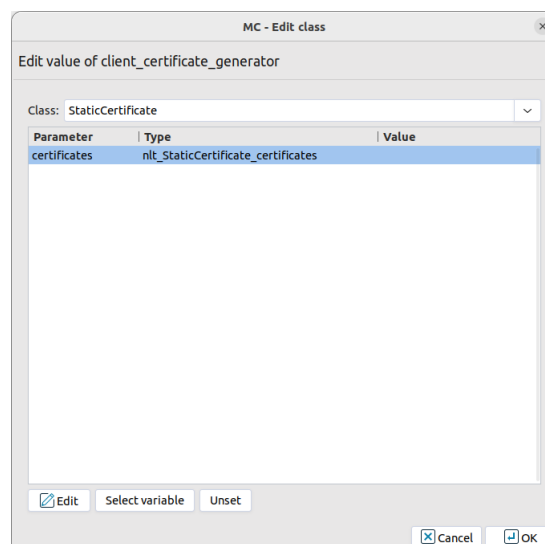

Figure 4. Creating a new Encryption policy

Python:

```
encryption=TwoSidedEncryption(
    client_certificate_generator=StaticCertificate(
        certificates=(
            Certificate.fromFile(
```

```
certificate_file_path="/etc/key.d/VMS_Engine/cert.pem",
                private_key=PrivateKey.fromFile(
                    "/etc/key.d/VMS_Engine/key.pem")
            ),
        )
    )
```

Step i. If the private key of the certificate is password-protected, double-click **private_key_password**, type the password, then click OK. Otherwise, click OK.

Step j. Disable mutual authentication. That way, Application-level Gateway will not request a certificate from the clients.
Double-click **client_verify**, select **Class > ClientNoneVerifier**, then click OK.
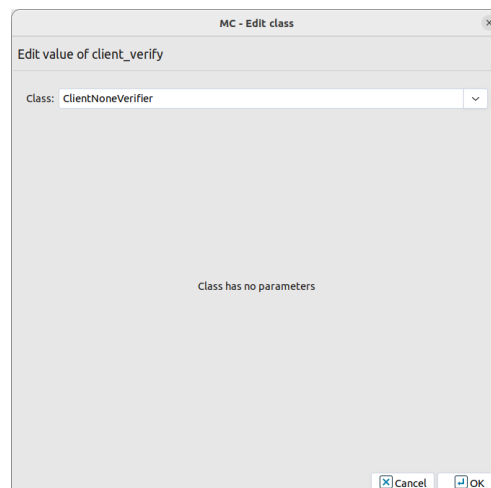
Figure 5. Disabling mutual authentication

Python:

```
encryption=TwoSidedEncryption(
    client_verify=None
)
```

Step k. Specify the directory containing the certificates of the trusted CAs. These settings determine which servers can the clients access: the clients will be able to connect only those servers via TLS which have certificate signed by one of these CAs (or a lower level CA in the CA chain).
Double-click **server_verify**, double-click **verify_ca_directory**, then type the path and name to the directory that stores the trusted CA certificates, for example, `/etc/ca.d/certs/.` Click OK.

Figure 6. Specifying trusted CAs

Python:

```
encryption=TwoSidedEncryption(
    server_verify=ServerCertificateVerifier(
        verify_ca_directory="/etc/ca.d/certs/"
    )
)
```

> **Note**
> CAs cannot be referenced directly, only the trusted group containing them. For details on managing trusted groups, see *Section 11.3.7.3, Managing trusted groups* in *Proxedo Network Security Suite 2 Administrator Guide*.

Step l. Specify the directory containing the CRLs of the trusted CAs.
Double-click **verify_crl_directory**, then type the path and name to the directory that stores the CRLs of the trusted CA certificates, for example, `/etc/ca.d/crls/`. Click OK.
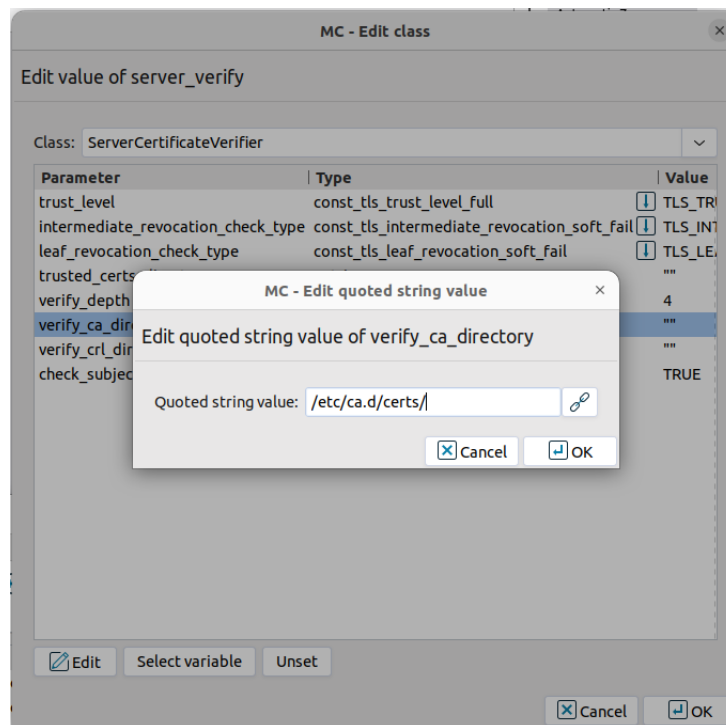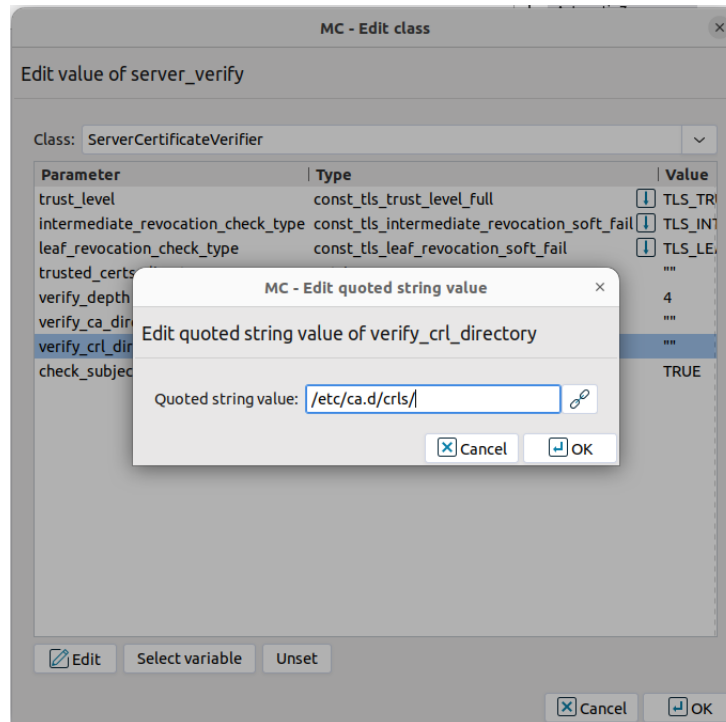
Figure 7. Specifying CRLs

Python:

```
encryption=TwoSidedEncryption(
    server_verify=ServerCertificateVerifier(
        verify_ca_directory="/etc/ca.d/certs/",
        verify_crl_directory="/etc/ca.d/crls/"
    )verify_
)
```

S t e p   *Optional Step*: The Common Name in the certificate of a server or webpage is usually
m.   its domain name or URL. By default, Application-level Gateway compares this Common
         Name to the actual domain name it receives from the server, and rejects the connection
         if they do not match. That way it is possible to detect several types of false certificates
         and prevent a number of phishing attacks. If this mode of operation interferes with
         your environment, and you cannot use certificates that have proper Common Names,
         disable this option.
         Double-click **server_verify > check_subject**, select *FALSE*, then click OK.

Python:

```
encryption=TwoSidedEncryption(
    server_verify=ServerCertificateVerifier(
        verify_ca_directory="/etc/ca.d/certs/",
        verify_crl_directory="/etc/ca.d/crls/",
        check_subject=FALSE
    )
)
```

Step n. *Optional Step*: Forbid the use of weak encryption algorithms to increase security. The related parameters can be set separately for the client and the server-side of Application-level Gateway, using the **client_tls_options** and **server_tls_options** parameters of the Encryption Policy. Disabling weak algorithms also eliminates the risk of downgrade attacks, where the attacker modifies the TLS session-initiation messages to force using weak encryption that can be easily decrypted by a third party.

> **Note**
> Certain outdated operating systems, or old browser applications do not properly support strong encryption algorithms. If your clients use such systems or applications, it might be required to permit weak encryption algorithms.

Step o. *Optional Step*: Enable untrusted certificates. Since a significant number of servers use self-signed certificates (with unverifiable trustworthiness), in certain situations it might be needed to permit access to servers that have untrusted certificates.

> **Note**
> When an untrusted certificate is accepted, the generated certificates will be signed with the untrusted CA during keybridge scenarios. For details on configuring keybridging, see *Procedure 2.2, Configuring keybridging* in *How to configure TLS proxying in PNS 2*

Double-click **server_verifier > trust_level**, click the drop-down menu and select `UNTRUSTED`, then click OK.

> **Note**
> When the **trust_level** value is *NONE*, even the invalid certificates are accepted and at the client side there is no client certificate request sent to the client.

Python:

```
encryption=TwoSidedEncryption(
    server_verify=ServerCertificateVerifier(
        trust_level=TLS_TRUST_LEVEL_UNTRUSTED
    )
)
```

Python:

The Encryption Policy configured in the previous steps is summarized in the following code snippet.

```
EncryptionPolicy(
    name="MyTLSEncryption",
    encryption=TwoSidedEncryption(
        client_verify=ClientNoneVerifier(),
        client_tls_options=ClientTLSOptions(),
        server_verify=ServerCertificateVerifier(
```

```
            trust_level=TLS_TRUST_LEVEL_FULL,
            intermediate_revocation_check_type =
                TLS_INTERMEDIATE_REVOCATION_SOFT_FAIL,
            leaf_revocation_check_type =
                TLS_LEAF_REVOCATION_SOFT_FAIL,
            trusted_certs_directory="",
            verify_depth=4,
            verify_ca_directory="/etc/ca.d/certs/",
            verify_crl_directory="/etc/ca.d/crls/",
            check_subject=TRUE
            ),
        server_tls_options=ServerTLSOptions(),
        client_certificate_generator=StaticCertificate(
            certificates=(
                Certificate.fromFile(
                    certificate_file_path=
                        "/etc/key.d/VMS_Engine/cert.chain.pem",
                    private_key=PrivateKey.fromFile(
                        "/etc/key.d/VMS_Engine/key.pem")),
            ))
    ))
```

Step 3. Select **PKI > Distribute Certificates**.
Note when managing PNS without MC, copy the certificates and CRLs to their respective directories. They are not updated automatically as in configurations managed by MC.

By performing the above steps, the proxy has been configured to use the specified certificate and its private key, and also the directory has been set that will store the certificates of the trusted CAs and their CRLs. Client authentication has also been disabled.

Step 4. Create a service that clients can use to access the Internet in a secure channel. This service will use the MyTLSEncryption Encryption Policy.

> Step a. Select **Services > New**, enter a name for the service (for example, intra_HTTPS_inter), then click OK.
>
> Step b. Select **Proxy class > Http > HttpProxy**.
>
> Step c. Select **Encryption > MyTLSEncryption**.
>
> Step d. Configure the other parameters of the service as neecessary for the environment, then click **OK**.
>
> Step e. Select **Firewall Rules > New > Service**, and select the service created in the previous step. For more details on creating firewall rules, see *Section 6.5, Configuring firewall rules* in *Proxedo Network Security Suite 2 Administrator Guide*.
>
> Step f. Configure the other parameters of the rule as necessary for the environment, then click **OK**.
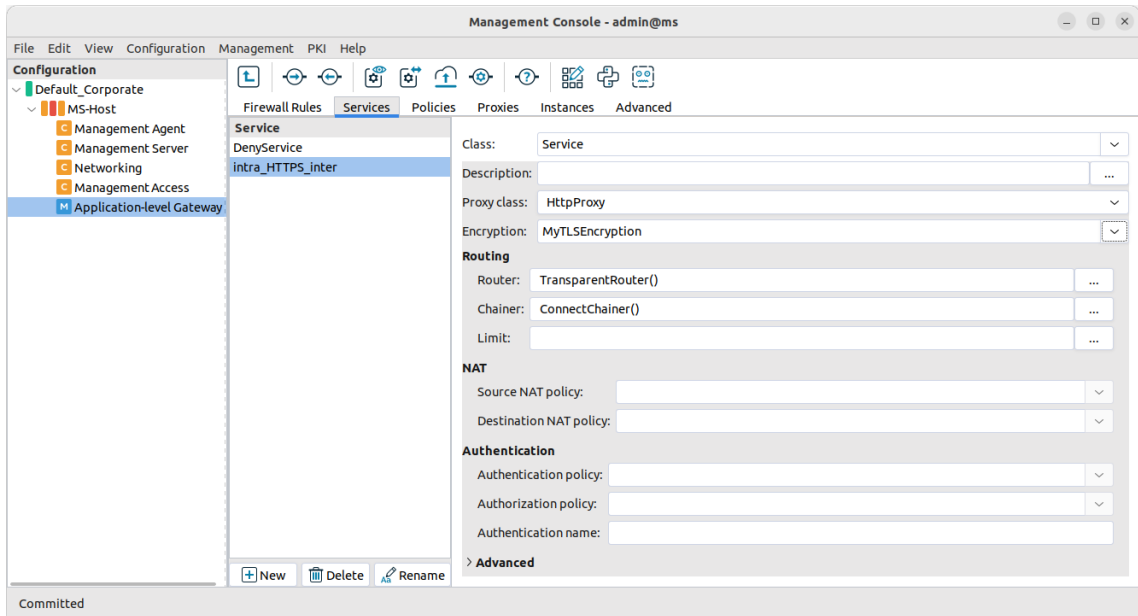
Figure 8. Creating a Service

Python:

```
def demo() :
    Service(
        name='demo/intra_HTTPS_inter',
        router=TransparentRouter(),
        chainer=ConnectChainer(),
        proxy_class=HttpProxy,
        max_instances=O,
        max_sessions=O,
        keepalive=V_KEEPALIVE_NONE,
        encryption_policy="MyTLSEncryption"
    )

    Rule(
        rule_id=3OO,
        src_subnet=('192.168.1.1/32', ),
        dst_zone=('internet', ),
        proto=6,
        service='demo/intra_HTTPS_inter'
    )
```

Step 5. Commit and upload the changes, then restart Application-level Gateway.
Expected result:

Every time a client connects to a server, Application-level Gateway checks the certificate of the server. If the signer CA is trusted, Application-level Gateway shows a trusted certificate to the client (browser or other application). If the certificate of the server is untrusted, Application-level Gateway shows an untrusted certificate to the client, giving a warning to the user. The user can then decide whether the certificate can be accepted or not.

2.2. Procedure – HTTPS and non-transparent proxying

Purpose:

The method described in *Procedure 2.1, Enabling TLS-encryption in the connection (p. 4)* can be used when the connections of the clients are proxied transparently. In the non-transparent case, use two HttpProxy classes. (A connection is non-transparent if the clients address the firewall host directly, and Application-level Gateway selects the target.)

Steps:

Step 1. Create and configure a transparent Http proxy to handle HTTPS connections as described in Steps 1-4 of *Procedure 2.1, Enabling TLS-encryption in the connection (p. 4)*. If a transparent HTTPS proxy has already been created and configured, skip this step.

Step 2. Navigate to **Application-level Gateway > Proxies**, and create a non-transparent HTTP proxy using the predefined `HttpProxyNonTransparent` proxy class. Name this new class, for example, `HttpSProxyNonTransparent`.

Step 3. Select this newly created proxy (for example, `HttpSProxyNonTransparent`) and add the `self.request` attribute to the **Changed config attributes** panel. To configure the `self.request` attribute, complete the following steps.

> Step a. Select the attribute and click **Edit**.
>
> Step b.   ■ To accept every request types, enter the * (asterisk) character, then click OK.
>
>   ■ Alternatively, you can add the request types you want to permit. It is recommended to enable the GET, POST, HEAD, and CONNECT requests.
>
> Step c. Click on the text in the **Type** field, then select `const_http_req_accept`.

Step 4. Add the `self.connect_proxy` attribute to the **Changed config attributes** panel, then click **Edit**. Select the proxy to be used for the HTTPS connections from the appearing list (for example, StrongHttpsProxy).

> **Note**
> This proxy is needed to handle the TLS data communicated in the plain-text nontransparent HTTP connection. If you do not want to examine that this traffic is indeed HTTP traffic, use a simple PlugProxy configured to handle TLS connections as well.

Python:

```python
class HttpSProxyNonTransparent(HttpProxyNonTransparent):
    def config(self):
        HttpProxyNonTransparent.config(self)
        self.connect_proxy=StrongHttpsProxy
        self.request["*"]=HTTP_REQ_ACCEPT
```

Step 5. Create a service that clients can use to access the Internet in a secure channel. This service will use the non-transparent Http proxy class (for example, `HttpSProxyNonTransparent`) created in Step 2.

Create a service that clients can use to access the Internet.

Step a. Select **Services** > **New**, and enter a name for the service (for example, `intra_HTTP_inter`).

Step b. Select **Proxy class > HttpSProxyNonTransparent**.

Step c. Select a **Router** for the service. Note the following points:

- When non-transparently proxying HTTP traffic without any parent proxy, the Service must use InbandRouter.

- If a parent proxy is used (that is, the clients connect to a web proxy like Squid through Application-level Gateway), DirectedRouter or InbandRouter can be used. InbandRouter can only be used if the `parent_proxy` and `parent_proxy_port` parameters are properly configured. If the firewall host is located network-transparently in front of the proxy server, even TransparentRouter can be used. For further details on Routers, see *Section 6.4.5, Routing — selecting routers and chainers* in *Proxedo Network Security Suite 2 Administrator Guide*.

Step d. Configure the other parameters of the service as needed for your environment, then click **OK**.

Step e. Select **Firewall Rules > New > Service**, and select the service created in the previous step.

> **Note**
> - If the clients connect directly to the firewall as a proxy, non-transparent service has to be used (using the same IP:port pair that is set on the clients).
> - If the firewall is located in front of the parent proxy used by the clients, a transparent listener has to be used, even though the proxy class used in the service is non-transparent.

Step f. Configure the other parameters of the rule as needed for your environment, then click **OK**.

Step 6. Commit and upload the changes, then restart Application-level Gateway.

3. One-sided TLS and HTTPS

This section shows how to solve various tasks and problems using one-sided HTTPS connections — sometimes encryption is only required on the client (or the server) side, for example, to decrease the load on the servers (TLS-offloading).

Note that certain issues and side-effects might need to be addressed when using one-sided TLS. These are discussed in *Section 3.2, Solving problems in one-sided HTTPS connections (p. 16)*.

3.1. Procedure – Configuring one-sided TLS

Purpose:

To disable encryption on one side of the connection for an existing Encryption Policy that is configured to handle HTTPS connections, complete the following steps.

> **Note**
> Obviously it is not possible to use keybridging together with one-sided TLS connections, but for a possible solution, see *Procedure 3.2.3, Transferring certificate information in one-sided HTTPS (p. 17)*.

Steps:

Step 1. Navigate to **Application-level Gateway > Proxies**, and select the proxy to be modified, or create a new one (for example, *OnesidedHttpsProxy*).

Step 2.
- To disable encryption on the client side, add the *self.tls.client_connection_security* parameter to the **Changed config attributes** panel, then set it to *const_tls_none*.
- To disable encryption on the server side, add the *self.tls.server_connection_security* parameter to the **Changed config attributes** panel, then set it to *const_tls_none*.

Python: Add one of the following lines to proxy:

```
self.tls.server_connection_security = TLS_NONE
self.tls.client_connection_security = TLS_NONE
```

Step 3. When PNS is used to protect the servers, you must deploy the certificate of the server (including its private key) to Application-level Gateway, so that Application-level Gateway can show the certificate to the clients that connect to the server. The proxy used in the connection must be configured to use this certificate when communicating with the clients. Complete the following steps.

Step a. Import the certificate of the server into MS, and set the firewall to be its owner host. For details, see *Procedure 11.3.8.6, Importing certificates* in *Proxedo Network Security Suite 2 Administrator Guide*.

Step b. Navigate to **Application-level Gateway > Proxies**, and select the proxy to be modified (for example, *OnesidedHttpsProxy*).

Step c. Select (or add, if not already present) the *self.tls.server_keypair_files* parameter, then click **Edit**.

Step d. A window showing the certificates available on the host appears. Select the certificate of the server.

> **Note**
> The list displays only the certificates where the firewall host is set as the owner host of the certificate (that is, both the certificate and its private key is available).

3.2. Solving problems in one-sided HTTPS connections

Using absolute URLs in one-sided TLS communication can pose problems and should be avoided whenever possible (relative links will work without any problem). A URL starts with *http://www...* on the non-encrypted side should be *https://www...* on the encrypted side. Certain applications can be configured to use HTTPS links in the HTTP requests instead of normal HTTP. If your environment uses an application that cannot be configured this way, see the following procedures for a possible solution.

3.2.1. Procedure – Microsoft Outlook Web Access

Purpose:

Outlook Web Access (OWA) can generate HTTPS links if it receives a special header in the request. This header notifies OWA that it is located behind an HTTPS frontend. Application-level Gateway can be configured to insert this header automatically. Complete the following steps.

Steps:

Step 1. Navigate to **Application-level Gateway > Proxies**, and derive an HttpProxy class (for example, *OWAHttpProxy*) from an Http proxy configured to handle one-sided TLS connections.

Step 2. Add the *self.request_header* attribute to the **Changed config attributes** panel.

Step 3. Select the newly added attribute, then select **Edit > New**.

Step 4. Enter *Front-End-Https* for the key name. This will be the name of the header inserted into the requests.

Step 5. Select the **Type** column, then select *type_http_hdr_insert*.

Step 6. Click **Edit**, then select the second row (the one with *qstring* in its Type column).

Step 7. Click **Edit**, then enter *on*. This will be the content of the inserted header.
Python:

```
self.request_header["Front-End-Https"]=(HTTP_HDR_INSERT, "on")
```

Step 8. Create a service that will use this new proxy (for example, *OWAHttpProxy*).

3.2.2. Procedure – Using stream editor in one-sided HTTPS

Purpose:

If a server application does not support secure connections, or uses absolute links and this behavior cannot be modified, Application-level Gateway can change the URLs in the traffic. This can be accomplished by stacking a sed (stream editor) Linux command (or if needed, a complete shell script) into the proxy.

Steps:

Step 1. Navigate to **Application-level Gateway > Proxies**, and select the HTTP proxy configured to handle one-sided TLS connections, or create a new one (for example, *HttpSedProxy*).

Step 2. Add *self.response_stack* attribute to the **Changed config attributes** panel, then click **Edit**. (You need the *self.response_stack* attribute, because the response of the server has to be changed.)

Step 3. Click **New**, then enter ***.

Step 4. Select the **Type** column, then select *type_http_stk_data*.

Step 5. Click **Edit**, select the row containing **vela_stack**, then click **Edit** again.

Step 6. Select **Stacking type > Program**, and enter the command to be executed: *sed -e 's|http://|https://|g'*
For details on the sed command, see the sed manual pages.

> **Note**
> The example sed command modifies all absolute links that appear in the traffic, that is, even links pointing to an external site will be modified. If possible, use at least the full domain name of the server in the sed command to avoid this problem (for example, *sed -e 's|http://www.example.com/|https://www.example.com|g'*). Be as specific as possible.

Python:

```
self.response_stack["*"]=(HTTP_STK_DATA, (V_STACK_PROGRAM, "sed -e
's|http://|https://|g'"))
```

Step 7. Create a service that will use this new proxy (for example, *HttpSedProxy*).

3.2.3. Procedure – Transferring certificate information in one-sided HTTPS

Purpose:

Client authentication in HTTPS is sometimes based on inspecting the certificate of the client. When PNS is protecting the server, keybridging can be used to transfer the information from the client certificate to the server. However, in one-sided TLS connections (for example, if the communication between PNS and the server is not encrypted), the server does not receive a TLS certificate, therefore user authentication must use another method. A simple solution to this problem is as follows:

PNS requests a certificate from the client the usual way, extracts the required information from the client certificate, then inserts this information into an HTTP header. The server then authenticates the user based on the information received in the HTTP header. To accomplish this, create a special HttpProxy using the **Class editor**.

Steps:

Step 1. Navigate to the **Application-level Gateway** MC component, and click on the **Class editor** icon in the menu bar.

Step 2. Click **New**, then select the **General** tab.

Step 3. Enter a name for the class (for example, *HttpsCertProxy*).

Step 4. Select **Parent class > OnesidedHttpsProxy**.

Step 5. Select **Class type > proxy**.

Step 6. Type or paste the following Python code. Based on these settings, the header of the proxy class will be generated automatically into the **Source code** field. You have to type the remaining part manually, or paste it from this document.

> ⚠ **Warning**
> The source code has to confirm to the syntax requirements of the Python language. Handle indentation with great care, since in Python indentation forms the blocks of code that are on the same level (many other languages use brackets for this purpose, for example, C uses curly brackets).

Python:

```
def config(self):
    OnesidedHttpsProxy.config(self)
        self.request_header["X-User-Certificate"]=\
        (HTTP_HDR_INSERT, self.tls.client_peer_certificate.subject)
```

Step 7. Click **OK** and **Close**.

Step 8. Create a service that will use this new proxy (for example, *HttpsCertProxy*), or modify an existing one.

4. Name-based virtual hosting and Server Name Indication (SNI)

Name-based virtual hosting is a method to provide services under multiple domain names from a single server (that is, several different domain names point to the same IP address). When receiving an HTTP request, the server decides which domain should receive the connection based on the "Host" header of the HTTP request. Each domain has its own certificate for secure connections. The problem is that the TLS connection is built before the client sends the first HTTP request: the server should show the certificate of the appropriate domain before receiving the HTTP header specifying the domain name. In earlier PNS versions, this situation was solved by either assigning a separate IP address to each domain name, or using IP aliasing. In Proxedo Network Security Suite 2 and later, Server Name Indication (SNI) can be used to overcome the problem.

If IP aliasing is not feasible for some reason, Application-level Gateway can be configured to overcome this problem by modifying the target address of the connection based on information arriving in the HTTP request. This solution requires a special Http proxy.

In the following example (*Procedure 4.1, Configuring Server Name Indication (SNI) (p. 19)*), Application-level Gateway determines the target address of the HTTPS connection based on the "Host" header. Note that any other information present in the HTTP traffic can be used for such purpose. For example, it is possible to direct different GET requests to different servers (for example, requests to *www.example.com* are directed to Server1, but *www.example.com/admin* is redirected to Server2). It is also possible to use different servers to serve the static and the dynamic contents (for example, by redirecting all requests to get jpg, gif, and similar files to a separate server).

> **Note**
> Although the connections can be redirected to different servers, only a single certificate can be shown to the clients, because Application-level Gateway must send the client-side certificate to the client before the client sends the first HTTP request. Consequently, Application-level Gateway cannot determine the target address at this stage.

4.1. Procedure – Configuring Server Name Indication (SNI)

Purpose:

To configure an HttpProxy in a name-based virtual hosting scenario that uses Server Name Indication (SNI) to determine the address of the target server, complete the following steps.

Steps:

Step 1.   Create and configure an Encryption Policy. Complete the following steps.

Step a. Navigate to the **Application-level Gateway** MC component of the firewall host.

Step b. Select **Policies > New**.

Step c. Enter a name into the **Policy name** field, for example, `MySNIEncryption`.

Step d. Select **Policy type > Encryption Policy**, then click OK.
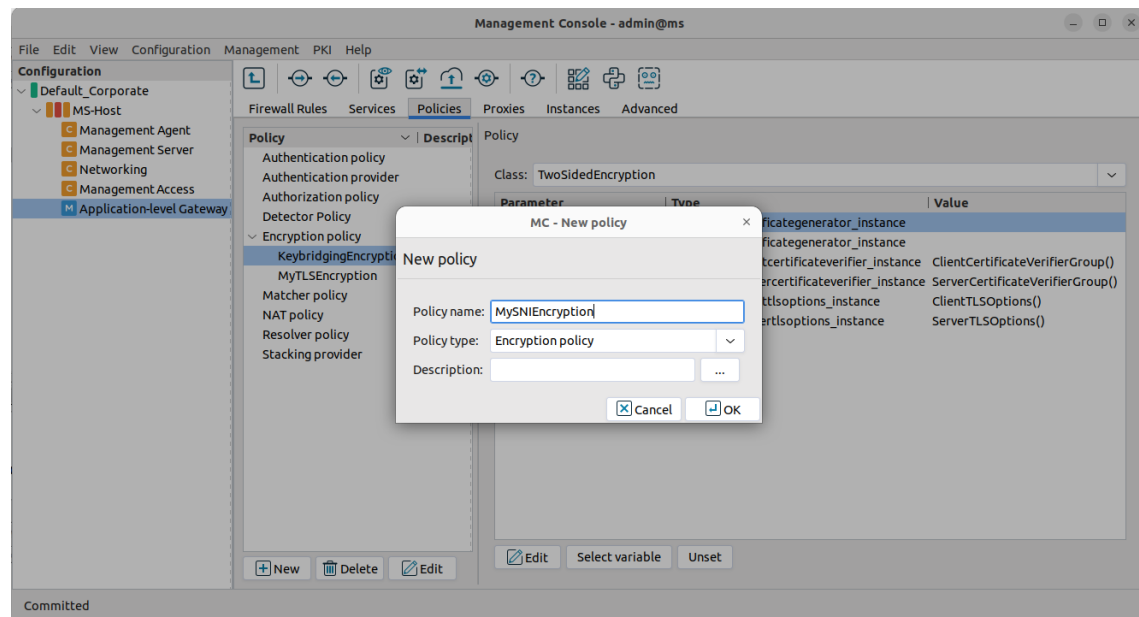
Figure 9. Creating an Encryption policy
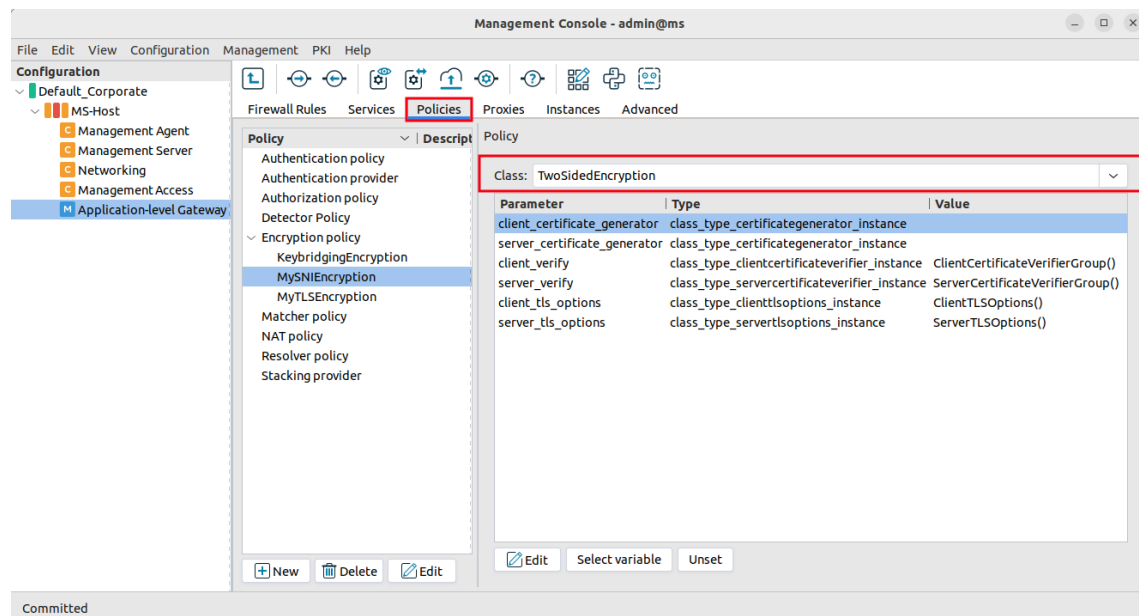
Step e. Select **Class > TwoSidedEncryption**.

Figure 10. Configuring the Encryption policy

Python:

```
EncryptionPolicy(
    name="MySNIEncryption",
    encryption=TwoSidedEncryption()
    )
```

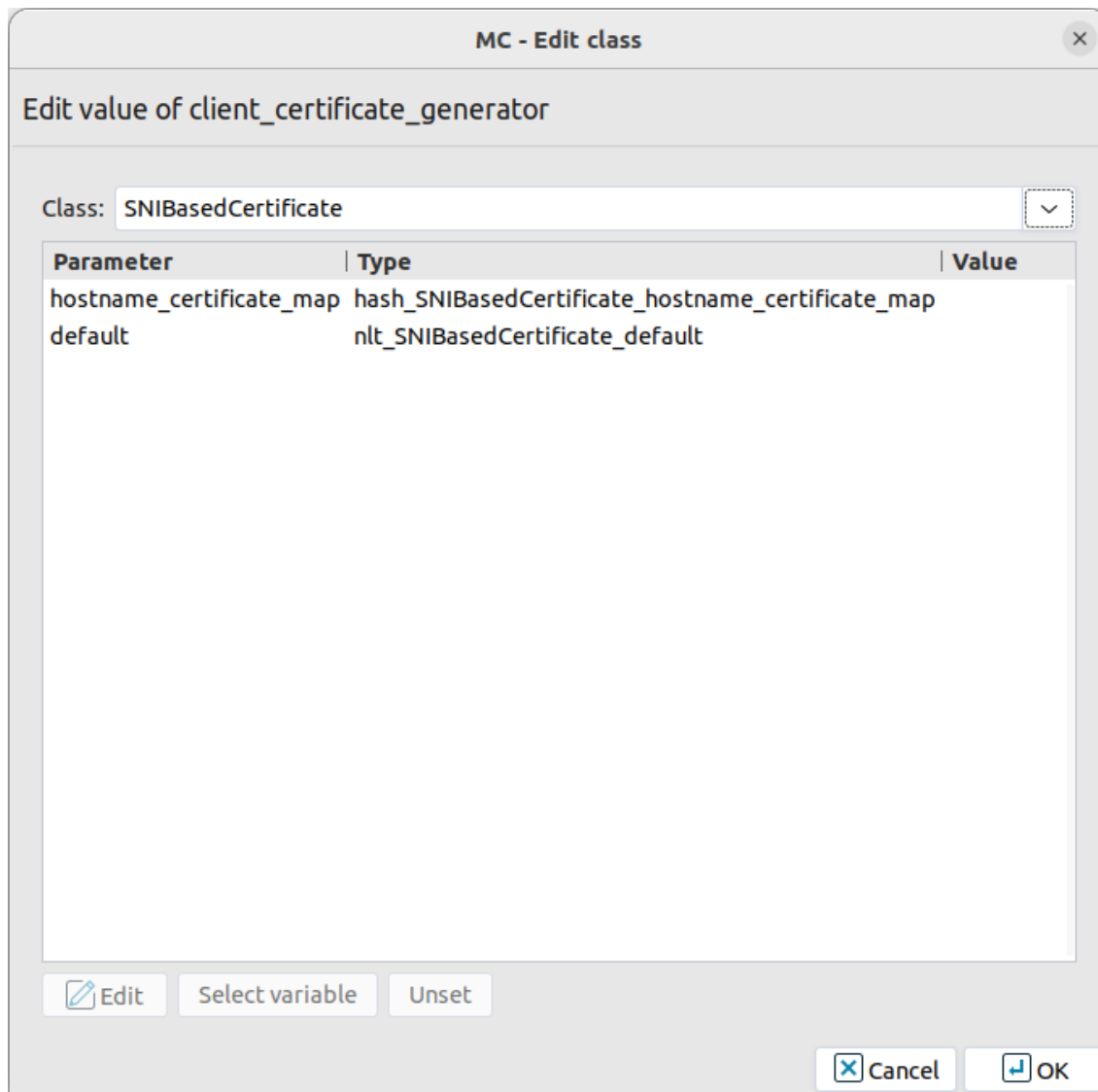Step 2. Double-click **client_certificate_generator**, then select **Class > SNIBasedCertificate**.

Figure 11. Configuring the certificate

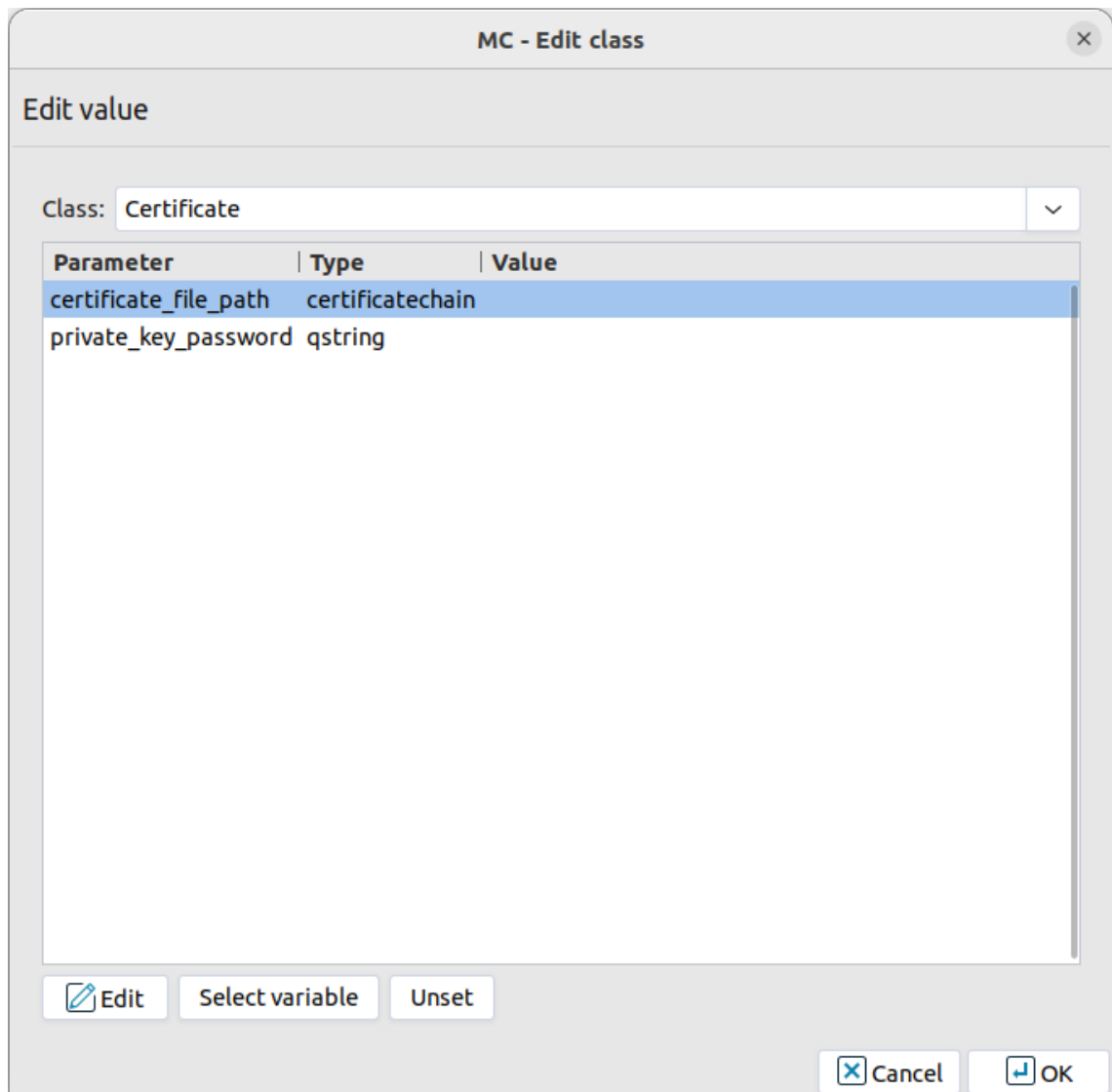Step 3. Double-click **default** and click **New** to add a certificate entry to a list of certificates.

Figure 12. Creating a new certificate entry

Step 4. Select the default certificate. Application-level Gateway will show this certificate to the clients when none of the other configured certificates match the client request.
Python:

```
encryption=TwoSidedEncryption(
    client_certificate_generator=SNIBasedCertificate(
        default=StaticCertificate(
            certificates=(
                Certificate.fromFile(

certificate_file_path="/etc/key.d/VMS_Engine/cert.pem",
                    private_key=PrivateKey.fromFile(
                        "/etc/key.d/VMS_Engine/key.pem")),
            )
        )
```

```
        )
    )
```

Step 5. Configure a mapping that describes to which hostname the list of certificates belongs to. For each certificate, configure a Matcher Policy. If this policy matches the domain name in the client SNI request, Application-level Gateway shows the associated certificate to the client. Any type of matcher policy can be used here, but in most scenarios only RegexpMatcher policies will be needed. (For details on Matcher Policies, see *Section 6.7.6, Matcher policies* in *Proxedo Network Security Suite 2 Administrator Guide*.)

The following example configures two matchers and two certificates, one for the myfirstdomain.example.com domain, one for the myseconddomain.example.com domain. Complete the following steps:

Step a. Double-click **hostname_certificate_map**, then click **New**.

Step b. Select **Class > RegexpMatcher**.

Step c. Click **Match > New**, then enter the domain name (for example, myfirstdomain.example.com) into the **Expression** field. Click **OK**.
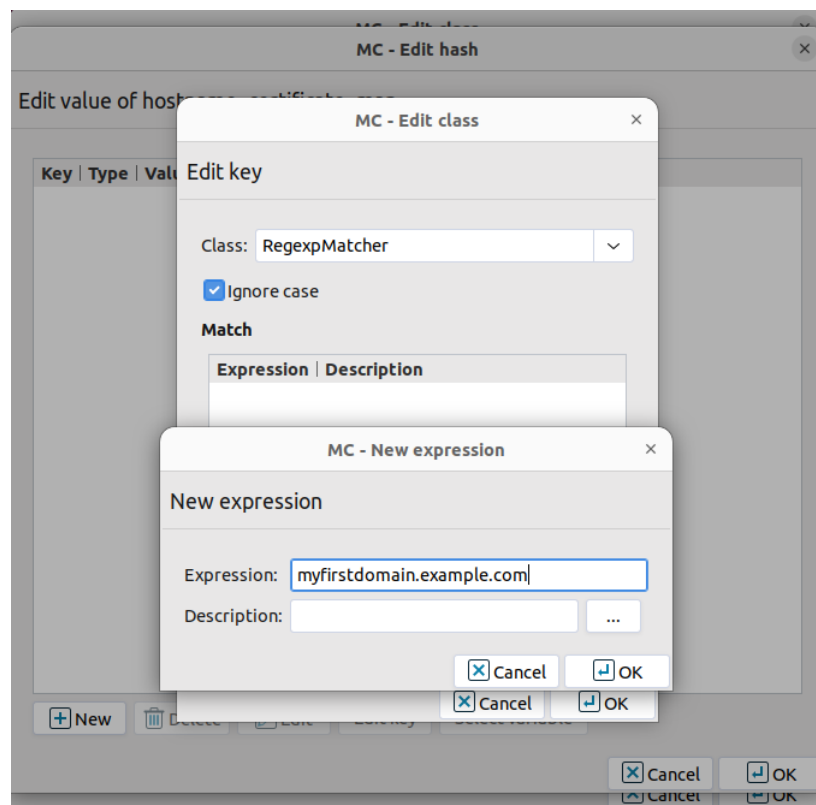

Figure 13. Configuring the hostname-certificate mapping

Step d. Click **Edit** and click **New** to add a certificate entry to a list of certificates.
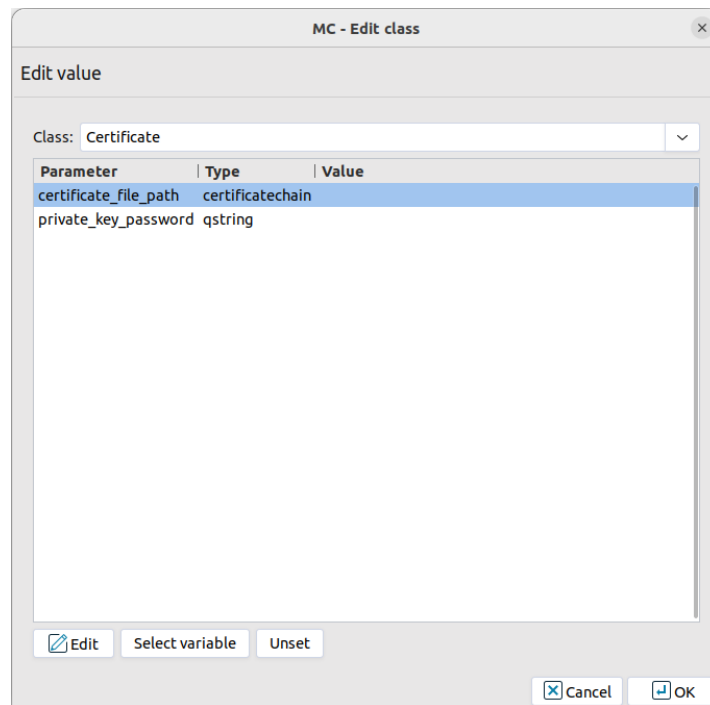
Figure 14. Creating a new certificate entry

Step e. Double-click **certificate_file_path** and select the certificate to show if a client tries to access the domain set in the previous step.

Step f. Click **Select**, then click **OK**.

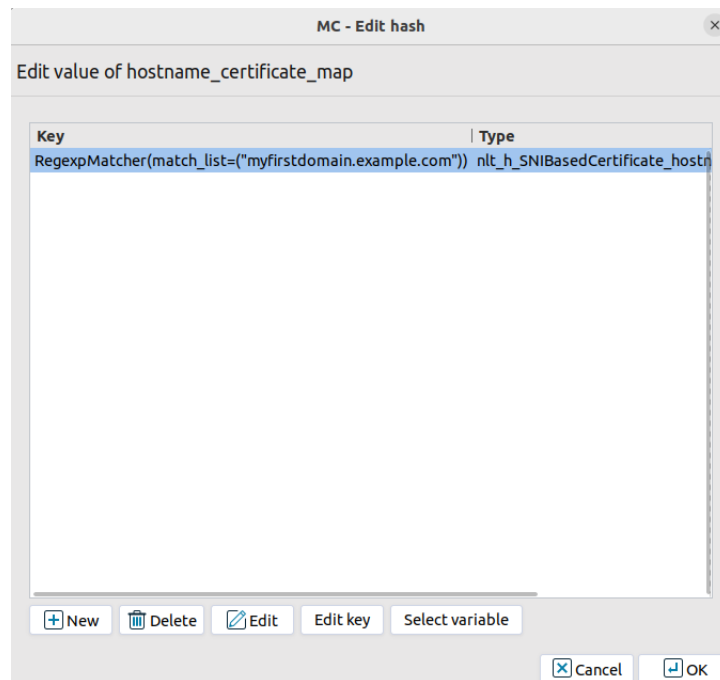Step g. Click OK to close the list editor.

Figure 15. Configuring the hostname-certificate mapping

Step h. Repeat from Step 'a' to Step 'e' for the `myseconddomain.example.com` domain and its respective certificate.

Python:

```
    encryption=TwoSidedEncryption(
        client_certificate_generator=SNIBasedCertificate(
            default=StaticCertificate(
                certificates=(
                    Certificate.fromFile(

certificate_file_path="/etc/key.d/VMS_Engine/cert.pem",
                        private_key=PrivateKey.fromFile(
                            "/etc/key.d/VMS_Engine/key.pem")),
                )),
            hostname_certificate_map={
                RegexpMatcher(
                    match_list=("myfirstdomain.example.com", )):
StaticCertificate(
                        certificates=(
                            Certificate.fromFile(

certificate_file_path="/etc/key.d/myfirstdomain/cert.pem",
                                private_key=PrivateKey.fromFile(
                                    "/etc/key.d/myfirstdomain/key.pem")),
                    )),
                RegexpMatcher(
                    match_list=("myseconddomain.example.com", )):
StaticCertificate(
```

```
                        certificates=(
                            Certificate.fromFile(

certificate_file_path="/etc/key.d/myseconddomain/cert.pem",
                                private_key=PrivateKey.fromFile(
                                    "/etc/key.d/myseconddomain/key.pem")),
                        ))
                },
            )
        )
```

Step 6. Configure the other options of the Encryption Policy as needed for your environment.

Step 7. Create a service and a firewall rule that uses this new Encryption Policy and an HttpProxy class.
Python:

```
def demo() :
    Service(
        name='demo/inter_HttpSNIService',
        router=TransparentRouter(),
        chainer=ConnectChainer(),
        proxy_class=HttpProxy,
        max_instances=O,
        max_sessions=O,
        keepalive=V_KEEPALIVE_NONE,
        encryption_policy="MySNIEncryption"
    )

    Rule(
        rule_id=3OO,
        src_subnet=('internet', ),
        dst_zone=('dmz', ),
        proto=6,
        service='demo/inter_HttpSNIService'
    )
```

5. Procedure – Enabling Windows update

Purpose:

To enable Windows update for the clients protected by the firewall, you have to import the certificate of the PNS CA that signs the certificates in keybridging into the client machines. To accomplish this, complete the following steps on the client hosts.

> **Note**
> An alternative to this solution is to disable TLS-proxying for the `v4.windowsupdate.microsoft.com` domain. This method is described in detail in the Technical Tutorial Proxying secure channels — the Secure Socket Layer. The advantage of the alternative method is that you do not need to modify the client hosts.

Prerequisite:

You will need the certificate of the PNS CA that signs the certificates in keybridging into the client machines. Export this certificate from MS, and make it available on your client hosts.

Steps:

Step 1.   Start the Microsoft Management Console (**Start Menu > Run application > MMC**).

Step 2.   Select **File > Add/Remove Snap-in**.

Step 3.   Click **Add**, then select **Certificates**.

Step 4.   Select **Computer account**, then click **Next**.

Step 5.   Select **Local computer** and click **Finish**. The Certificates module has been added to the Console Root tree.

Step 6.   Expand the **Certificates** node, then expand the **Trusted Root Certification Authorities** node. Right-click on the `Certificates` node, select **All Tasks**, then click **Import**.

Step 7.   Click **Next** on the Welcome to the Certificate Import Wizard page. On the File to Import page, click **Browse**, and locate the certificate of the PNS CA to be imported.

Step 8.   On the Certificate Store page, accept the default setting (**Place all certificates in the following store**), click **Next**, then **Finish**.

> **Note**
> Application-level Gateway must be able to verify the certificates of the Windows Update servers. To accomplish this, the certificates of the certificate authorities (CAs) issuing the certificates of the Windows update servers have to be imported into Application-level Gateway, if not already present. The following certificates have to be imported:
> - Microsoft Secure Server Authority
> - Microsoft Internet Authority
> - GTE CyberTrust Global Root

6. Python code summary

When configured according to this tutorial, the `policy.py` file of Application-level Gateway should look something like this:

Configuring HTTPS proxying:

```
class HttpsProxy(HttpProxy):
    def config(self):
        HttpProxy.config(self)
        self.tls.client_keypair_files=\
            ("/etc/key.d/Certificate_for_TLS_proxying/cert.pem",\
            "/etc/key.d/Certificate_for_TLS_proxying/key.pem")
        self.tls.client_verify_type=TLS_VERIFY_NONE
        self.tls.client_connection_security = TLS_FORCE_TLS
        self.tls.server_connection_security = TLS_FORCE_TLS
        self.tls.server_cagroup_directories= \
            ("/etc/ca.d/groups/VMS_Trusted_CA/certs/",\
             "/etc/ca.d/groups/VMS_Trusted_CA/crls/")
```

Nontransparent version:

```
class HttpSNonTransparent(HttpProxyNonTransparent):
    def config(self):
        HttpProxyNonTransparent.config(self)
        self.connect_proxy= HttpsProxy
        self.request["GET"]=HTTP_REQ_ACCEPT
        self.request["POST"]=HTTP_REQ_ACCEPT
        self.request["HEAD"]=HTTP_REQ_ACCEPT
        self.request["CONNECT"]=HTTP_REQ_ACCEPT
```

One-sided HTTPS and Microsoft Outlook Web Access:

```
class OnesidedHttpsProxy(HttpsProxy):
    def config(self):
        HttpsProxy.config(self)
        self.tls.server_connection_security=TLS_NONE
        self.tls.server_keypair_files = \
            ("/etc/key.d/Sample Certificate/cert.pem",\
             "/etc/key.d/Sample Certificate/key.pem")
        self.stack_proxy=(V_STACK_PROXY, OWAHttpProxy)

class OWAHttpProxy(HttpProxy):
    def config(self):
        HttpProxy.config(self)
        self.request_header["Front-End-Https"]=(HTTP_HDR_INSERT, "on")
```

HTTP Proxy using stream editor

```
class HttpSedProxy(OnesidedHttpsProxy):
    def config(self):
        OnesidedHttpsProxy.config(self)
        self.response_stack["*"]=(HTTP_STK_DATA, (V_STACK_PROGRAM, "sed -e
's|http://|https://|g'"))
```

Transferring certificate information in an HTTP header

```
class HttpsCertProxy(OnesidedHttpsProxy):
    def config(self):
        OnesidedHttpsProxy.config(self)
        self.request_header["X-User-Certificate"]=(HTTP_HDR_INSERT,
self.tls.client_peer_certificate.subject)
```

Name-based virtual hosting and sidestacking:

```
class HttpProxyTargetByHostHeader(HttpProxy):
    def config(self):
        HttpProxy.config(self)
        self.request_header["Host"]=(HTTP_HDR_POLICY, self.TargetByHostHeader)
        self.tls.client_connection_security=TLS_FORCE_TLS
        self.tls.server_connection_security=TLS_NONE
        self.tls.server_keypair_files = \
            ("/etc/key.d/Sample Certificate/cert.pem",\
             "/etc/key.d/Sample Certificate/key.pem")
    def TargetByHostHeader(self, name, value):
        if (value == "example.com"):
          self.session.setServer(SockAddrInet("192.168.0.1", 80))
          return HTTP_HDR_ACCEPT
        elif (value == "example2.com"):
          self.session.setServer(SockAddrInet("192.168.0.2", 80))
          return HTTP_HDR_ACCEPT
        return HTTP_HDR_ABORT
```

7. Summary

This tutorial has shown how to configure PNS to proxy HTTPS traffic, including scenarios where only one side of the traffic is encrypted. Although these examples are relatively simple, they provide a solid base from which more complex configurations can be built — just as the security policy of your organization requires it.

The latest version is always available at the BalaSys *Documentation Page*.