paloalto TECHDOCS

PAN-OS® and Panorama™ API Usage Guide

Version 11.0

docs.paloaltonetworks.com

Contact Information

Corporate Headquarters:
Palo Alto Networks
3000 Tannery Way
Santa Clara, CA 95054
www.paloaltonetworks.com/company/contact-support

About the Documentation

- For the most recent version of this guide or for access to related documentation, visit the Technical Documentation portal docs.paloaltonetworks.com.
- To search for a specific topic, go to our search page docs.paloaltonetworks.com/search.html.
- Have feedback or questions for us? Leave a comment on any page in the portal, or write to us at documentation@paloaltonetworks.com.

Copyright

Last Revised

December 6, 2023

Table of Contents

About the PAN-OS API

The PAN-OS® and Panorama™ API allows you to manage firewalls and Panorama through a third-party service, application, or script. The firewalls and Panorama support two types of API—XML API and REST API.

The XML API uses a tree of XML nodes to map firewall or Panorama functionality. To make an API request, you must specify the XPath (XML Path Language) to the XML node that corresponds to a specific setting or action. XPath allows you to navigate through the hierarchical XML tree structure for firewalls and Panorama. To get started, see:

- PAN-OS XML API Components
- Structure of a PAN-OS XML API Request

You can use the REST API to Create, Update, Read, Delete (CRUD) Objects and Policies on the firewalls; you can access the REST API directly on the firewall or use Panorama to perform these operation on policies and objects from a central location and push them to the managed firewalls. To get started, see Access the PAN-OS REST API.

Because PAN-OS API functionality mirrors that of both the web interface and the CLI, you should familiarize yourself with both. Reading relevant portions of the PAN-OS Administrator's Guide will help you get a better understanding of firewall functionalities that you can access using the API. You should also be knowledgeable about web service APIs, HTTP, XML, and XPath.

PAN-OS XML API Components

Use the PAN-OS XML API when you want to automate tasks you need to perform, such as:

- Create, update, and modify firewall and Panorama configurations.
- Execute operational mode commands, such as restart the system or validate configurations.
- Retrieve reports.
- Manage users through User-ID.
- Update dynamic objects without having to modify or commit new configurations.

The PAN-OS XML API offers a number of components to automate access and configuration of Palo Alto Networks firewalls and Panorama.

Feature	Description
Full access to PAN-OS functionality	The PAN-OS XML API allows you to access almost all of the functionality normally provided through the firewall web interface and CLI.
Secure authentication and access using API key and admin roles	Use your administrative username and password to generate an API key to authenticate API calls. Granular roles allow you to grant API access to specific functionality including reports, logs, and operational mode commands.
Options to view XML syntax through API browser, CLI and web interface debug mode	To explore all various functions of the API, you can use the API browser through the firewall web interface. You can also enable debug mode through the CLI to see the API equivalent of CLI commands.

When multiple login or logout events are generated at the same time, make sure to follow these guidelines to ensure optimal firewall performance:

- Design your application to queue events and perform batch API updates instead of sending single event or mapping updates.
- Limit the number of concurrent API calls to five. The suggested limit ensures that there is no performance impact to the firewall web interface as the management plane web server handles requests from both the API and the web interface. Limits may vary depending on the type of request.

To learn about the PAN-OS REST API, see PAN-OS REST API.

To learn about changes to the latest version of CLI commands that affect corresponding PAN-OS XML API requests, see the PAN-OS CLI Quick Start.

Structure of a PAN-OS XML API Request

A PAN-OS XML API request typically comprises a number of parameters, as shown in the example below:

```
curl -X POST 'https://firewall/api?
type=<type>&action=<action>&xpath=<xpath>&key=<apikey>"
```

- API key (**key=**): The API key allows you to authenticate yourself to the API when making requests. Learn about API Authentication and Security and how to Get Your API Key.
- Request type (**type=**): Because the XML API allows you to perform a wide array of requests, you must first specify the type of request you want, ranging from configuration to operation, importing to exporting, and from reports to user ID. Learn more about Request Types.
- Action (**action=**): When the request type is**config** (configuration) or**op** (operational mode command), you must also specify an associated action, such as**edit**,**delete**, or**move**. Learn more about Configuration Actions.
- XML and XPath elements (**xpath=** or**cmd=**): When using configuration or operational mode commands on the firewall, you include only the XML or the XPath that specifies the XML node. Learn more about XML and XPath and XPath Node Selection.

To make requests to the PAN-OS XML API, you can use the GET and POST methods.

Use a GET request when the query size is less than 2K and you want to pass strings in the Request URL. When using the GET method, append the query string to the request URL as a URL-encoded parameter string:

```
GET /api/?type=keygen&user=<username>&password=<password>
```

Use a POST request when you are sending large amounts of form data (the request size is between 2K to 5MB; limit the request size to 5MB) or when you are passing non-ASCII characters. Some API requests, such as importing files, require POST. When using the POST method, pass the parameters in the request body. In this example, the request body includes the login credentials:

```
POST /api/ HTTP/1.1

Content-Type: application/x-www-form-urlencoded

password=<password>&user=<username>&type=keygen
```

If you want to learn about the PAN-OS REST API structure, see PAN-OS REST API Request and Response Structure.

API Authentication and Security

To use the API (XML or REST), you must enable API access for your administrators and get your API key. By default, the firewall and Panorama support API requests over HTTPS. To make API request over HTTP, you must configure an interface management profile.

To authenticate your API request to the firewall or Panorama, provide the API key in any of the following ways:

- Use the custom HTTP header,**X-PAN-KEY: <key>** to include the API key in the HTTP header.
- For the XML API, include the API key as a query parameter in the HTTP request URL.
- Use Basic Authentication to pass the admin credentials as**username:password** with Base64 encoding in an Authorization header field.

```
Authorization: Basic amJPbLxpbw9UaTpXb3JrKjIwMDA=
```

As a best practice:

- Set an API key lifetime to enforce key rotation; you can also revoke all API keys to protect from accidental exposure.
- Use a POST request for any call that may contain sensitive information.

You cannot use basic authentication when you Get Your API Key.

To enforce key rotation set an API key lifetime; you can also revoke all API keys to protect from accidental exposure.

XML and XPath

The PAN-OS XML API uses XML for both requests and responses. When making requests, construct an HTTPS GET or POST request with the correct type and action along with the correct XPath. Here is an example API request:

```
curl -X POST 'https://firewall/api?
type=config&action=show&key=<APIkey>&xpath=/config/devices/entry/
vsys/entry/rulebase/security"
```

Ensure you replace variables such as <hostname> and <APIkey> with the IP address or hostname of your firewall or Panorama and API key, respectively.

When making configuration requests (**type=config**), you can use XPath, a syntax for selecting nodes from within an XML document. Use the XPath to isolate and modify portions of your configuration. The XML configuration within PAN-OS uses four different types of nodes as shown here:

```
<users>
  <entry name="admin">
    <permissions>
      <role-based>
        <superuser>yes</superuser>
      </role-based>
    </permissions>
  </entry>
```



```
  <entry name="guest">
    <permissions>
      <role-based>
        <custom>
          <profile>NewUser</profile>
        </custom>
      </role-based>
    </permissions>
  </entry>
</users>
```

- Root nodes are top-level nodes with no parent. Requesting the root node returns all child elements.
- Element nodes represent containers of information. Element nodes can contain other element nodes or simply act as a container of information. Example:**<permissions></permissions>**
- Attribute nodes are nodes that contain name/value pairs. Example:**<entry name="admin"></entry>**
- Text nodes contain plain text. Example:**<superuser>yes</superuser>**

Explore the API with the API browser, CLI, or debug console to learn how to construct XML requests.

XPath Node Selection

There are various ways to specify the XPath for an XML node in an API request. The simplest is to use the location path of the resource. For example, to select all users within your management configuration, use the following path:

```
/config/mgt-config/users
```

The above path specifies the following XML node that includes all users:

```
<users>
  <entry name="admin">
    <permissions>
      <role-based>
        <superuser>yes</superuser>
      </role-based>
    </permissions>
  </entry>
  <entry name="guest">
    <permissions>
      <role-based>
        <custom>
          <profile>NewUser</profile>
        </custom>
      </role-based>
    </permissions>
  </entry>
</users>
```

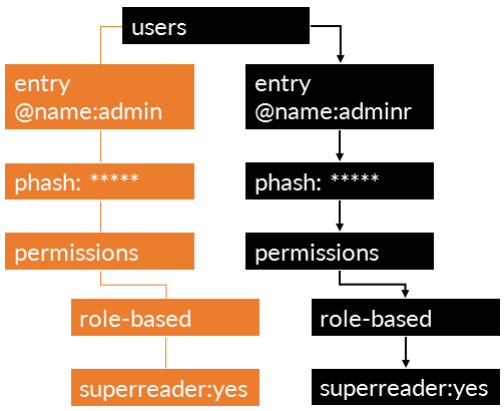
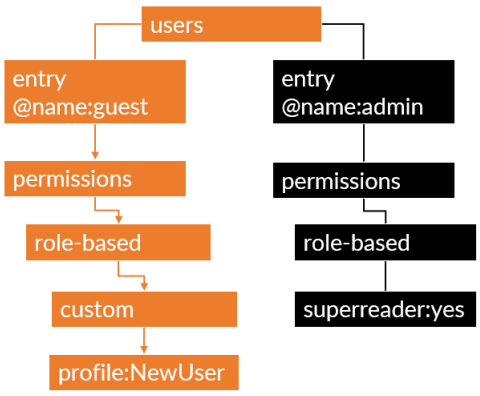

Targeting multiple nodes in an XPath using nested elements results in a successful command, but will not update all of the nodes. To update each node, send the configuration to each node using multiple successive calls. For example:

```
/entry[@name='TEST_IKE_PAN']/protocol/ikev1/
dpd&element=<enable>yes</enable></dpd></
ikev1><version>ikev2-preferred</version></protocol>&/
ikev2&element=<ike-crypto-profile>default</ike-crypto-
profile></ikev2>&/peer-address&element=<ip>1.2.3.4</ip>
```

To successfully update each node, target each node individually, for example:

```
entry[@name='TEST_IKE_PAN']/peer-
address&element=<ip>1.2.3.4</ip>
```

Another method for selecting the XPath for an XML node is to select the specific node, such as the**superuser** or**NewUser** node within the node shown above. Use XPath syntax similar to the following to drill-down and select a specific node:

XML Node	XPath Syntax
	**/config/mgt-config/users/ entry/permissions/role-based/ superuser[text()='yes']**
	**/config/mgt-config/users/entry/ permissions/role-based/custom/ profile[text()='NewUser']**

Get Started with the PAN-OS XML API

To use the PAN-OS XML API, first use your admin credentials to get an API key through the `keygen` command type. You can then use the API key to test a simple call.

- Enable API Access
- Get Your API Key
- Make Your First API Call
- Explore the API
- PAN-OS XML API Error Codes

This guide exercises API requests using cURL commands. However, you can use other API tools such as Postman and RESTClient to make API requests. By default, PAN-OS uses a self-signed certificate, so you will need to use -k parameter with cURL requests. Alternatively, you must replace the self-signed certificate with one from a known certificate authority. If you have an internal certificate authority, generate your own certificate and install it on the firewall.

Enable API Access

The API supports the following types of Administrators and Admin roles:

- Dynamic roles: Superuser, Superuser (readonly), Device admin, Device admin (readonly), Vsys admin, Vsys admin (readonly)
- Role-based Admins: Device, Vsys, Panorama.

Admin Role profiles enable or disable features on the management interfaces of the firewall or Panorama, XML API, web interface, and CLI. For more details on Administrative Roles, see Configure an Admin Role Profile.

As a best practice, set up a separate admin account for XML API access.

STEP 1 | Select an Admin Role profile.

Go to **Device** > **Admin Roles** and select or create an admin role.

STEP 2 | Select features available to the admin role.

1. Select the **XML API** tab.
2. Enable or disable XML API features from the list, such as **Report**, **Log**, and **Configuration**.
3. Select **OK** to confirm your change.

STEP 3 | Assign the admin role to an administrator account.

See Configure an Administrative Account.

Get Your API Key

To use the API, you must generate the API key required for authenticating API calls.

Then, when you use this API key in your request, you can either provide the URL encoded API key in the request URL, or use the custom X-PAN-KEY: <key> parameter to add the key as a name-value pair in the HTTP header.

If you have an existing key and generate another key for the same user, all existing sessions will end for the user and previous API sessions will be deleted. If the cookie for the request doesn't exist but you make subsequent requests, configuration logs will show the user as unknown.

STEP 1 | To generate an API key, make a POST request to the firewall's hostname or IP addresses using the administrative credentials and**type=keygen**:

```
curl -H "Content-Type: application/x-www-form-
urlencoded" -X POST https://firewall/api/?type=keygen -d
 'user=<user>&password=<password>'
```

A successful API call returns `status="success"` along with the API key within the `key` element:

```
<response status="success">
  <result>
    <key>gJlQWE56987nBxIqyfa62sZeRtYuIo2BgzEA9UOnlZBhU==</key>
  </result>
</response>
```

This is an example API Key, when you retrieve your API Key, use the key in its entirety, including any symbols such as equal signs.

STEP 2 | (Optional) Revoke API keys.

You can revoke all currently valid API keys, in the event one or more keys are compromised. To change an API key associated with an administrator account change the password associated with the administrator account. API keys that were generated before you expired all keys, or a key that was created using the previous credentials will no longer be valid.

If you use Panorama to manage your firewalls, Panorama and all of the firewalls that it manages must have the same master key.

Authenticate Your API Requests

Palo Alto Networks encourages you to authenticate your API requests by including a basic authentication token in the header of your requests. The basic authentication header can be used to authenticate both XML and REST API requests.

STEP 1 | Convert your user name and password to Base64 format.

Example: `username:password` converts to dXNlcm5hbWU6cGFzc3dvcmQ=

STEP 2 | When making a request to the firewall, include the base64 converted token in the header preceded by `Authorization: Basic`

Example:

```
curl -X POST 'https://<firewall>/api?
&type=config&action=get&xpath=/config/devices/entry[@name=
%27localhost.localdomain%27]/network/interface/ethernet' -H
 'Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ='
```

STEP 3 | Include the header in each of the subsequent requests to the firewall.

Make Your First API Call

Get Your API Key to make your first call to the PAN-OS XML API. Make sure to URL encode the request parameters in the HTTP request.

The API Docs use a number of general conventions and should not be copy and pasted verbatim. Adjust the call to your specific firewall before making the request.

Variable	Replace With
<firewall>	The IP address of the firewall or Panorama appliance you intend to target with your request.
apikey	The unique API key you generate.

All the query strings in Get requests must be a URL-Encoded parameter string. If you use a space in the URL-Encoded request, you must include either a plus sign or %20 to replace the space.

If you have trouble replicating any of the API requests in our documentation as a first step, Use the API Browser to build your requests.

STEP 1 | Make a cURL call to get system information, which returns the IP address, hostname, and model of your firewall. Be sure to include the API key:

```
curl -X POST 'https://<firewall>/api?
type=op&cmd=<show><system><info></info></system></show>'
```

When you make your API calls, as an alternative to providing the URL encoded API key in the request URL, you can use the custom X-PAN-KEY: <key> parameter to add the key as a name value pair in the HTTP header. For example, ***curl -H "X-PAN-KEY: LU234T02234565s2Z1FtZWFyWXJOSTdk1234565234565=" -k 'https://firewall_IP/api/?type=op&cmd=<show><system><info></info></system></show>'***

STEP 2 | Confirm that the response to the above request looks similar to this:

```
<response status="success">
  <result>
    <system>
      <hostname>PA-3050-A</hostname>
      <ip-address>10.2.3.4</ip-address>
      <public-ip-address>unknown</public-ip-address>
      <netmask>255.255.252.0</netmask>
      <default-gateway>10.2.3.1</default-gateway>
      <is-dhcp>no</is-dhcp>
      <ipv6-address>unknown</ipv6-address>
      <ipv6-link-local-address>c123::21b:ffff:feff:c1234/64</ipv6-
link-local-address>
      <ipv6-default-gateway/>
      <mac-address>00:00:00:ff:c7:00</mac-address>
```



```
      <time>Tue Jan  8 16:22:56 2019</time>
      <uptime>0 days, 18:28:38</uptime>
      <devicename>PA-3050-A</devicename>
      <family>3000</family>
      <model>PA-3050</model>
      <serial>0017010.2529</serial>
      <cloud-mode>non-cloud</cloud-mode>
      <sw-version>9.0.0-b36</sw-version>
      <global-protect-client-package-version>0.0.0</global-protect-
client-package-version>
      <app-version>8111-5239</app-version>
      <app-release-date>2019/01/07 15:51:30 PST</app-release-date>
      <av-version>3328-3783</av-version>
      <av-release-date>2019/01/07 11:22:02 PST</av-release-date>
      <threat-version>8111-5239</threat-version>
      <threat-release-date>2019/01/07 15:51:30 PST</threat-release-
date>
      <wf-private-version>0</wf-private-version>
      <wf-private-release-date>unknown</wf-private-release-date>
      <url-db>paloaltonetworks</url-db>
      <wildfire-version>0</wildfire-version>
      <wildfire-release-date/>
      <url-filtering-version>2019010.2.00005</url-filtering-
version>
      <global-protect-datafile-version>unknown</global-protect-
datafile-version>
      <global-protect-datafile-release-date>unknown</global-
protect-datafile-release-date>
      <global-protect-clientless-vpn-version>0</global-protect-
clientless-vpn-version>
      <global-protect-clientless-vpn-release-date/>
      <logdb-version>9.0.10</logdb-version>
      <platform-family>3000</platform-family>
      <vpn-disable-mode>off</vpn-disable-mode>
      <multi-vsys>on</multi-vsys>
      <operational-mode>normal</operational-mode>
    </system>
  </result>
</response>
```


Explore the API

There are several ways you can explore the API and learn how to construct your XML requests:

- Use the API Browser
- Use the CLI to Find XML API Syntax
- Use the Web Interface to Find XML API Syntax

Use the API Browser

Each firewall and Panorama provides an API browser that is accessible from your web browser. The API browser lets you navigate through and view the corresponding XPath and API URL.

STEP 1 | Launch the web interface.

1. Use a web browser to navigate to the actual FQDN or IP address of your firewall:
   `https://<firewall>/`
2. Log in with your administrator credentials when prompted to log in to the web interface.

STEP 2 | Launch the API Browser.

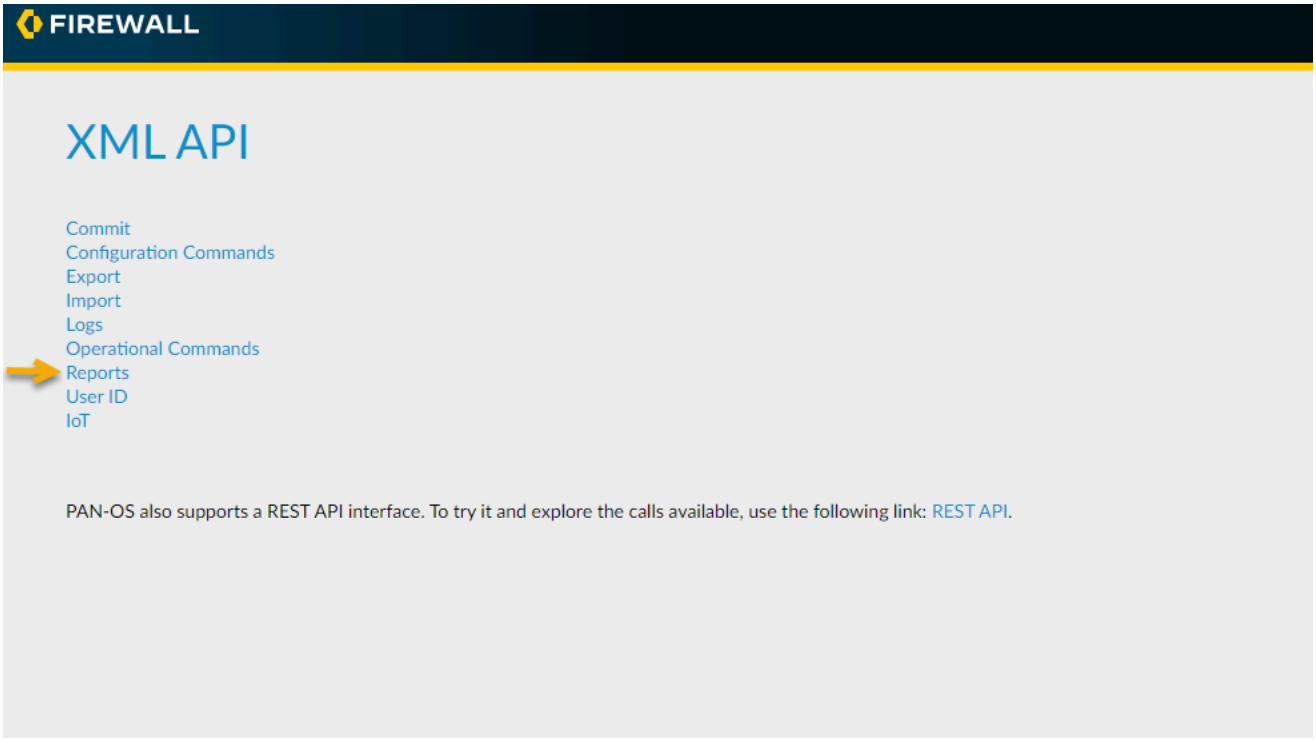
Go to the API browser URL on your firewall:

`https://<firewall>/api`

STEP 3 | Drill-down to a request.

When you first open the API browser, the available Request Types display.

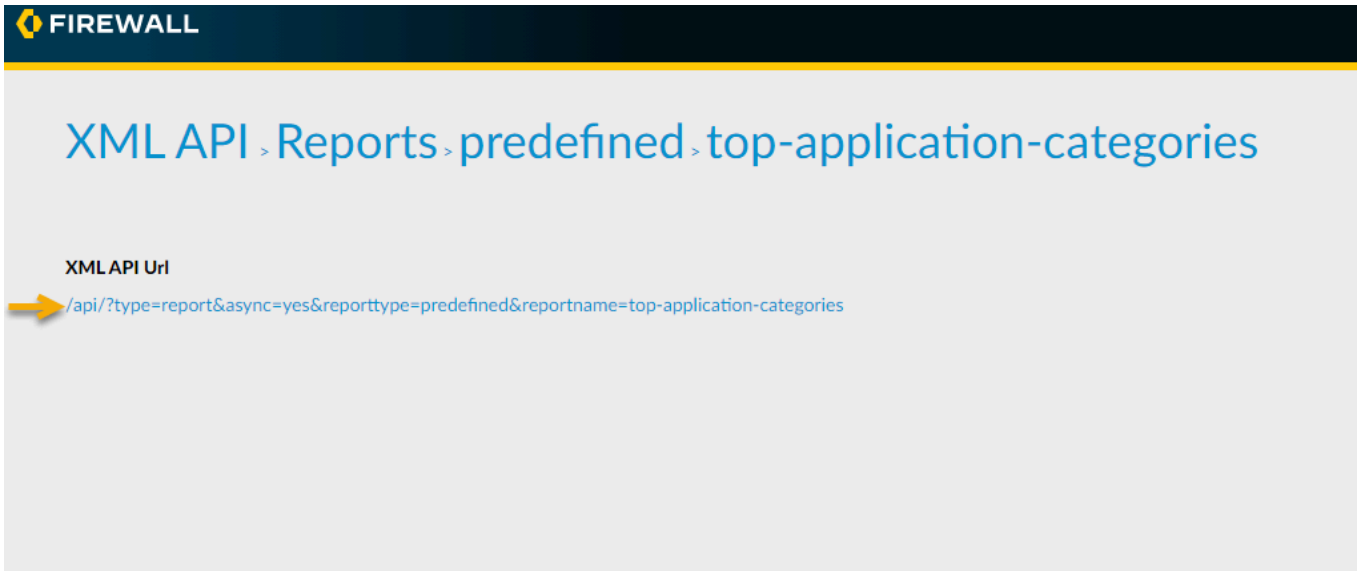
1. Select one of the request types to drill down to the next level of the XPath. Let's start with Configuration Commands, which equates to **`type=report`**:



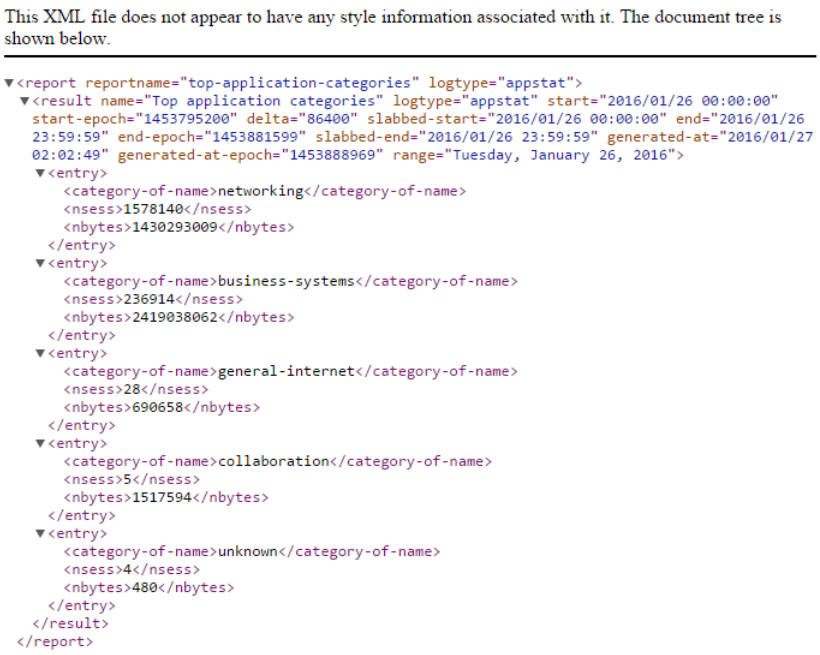
2. Drill down further until you select a request that you want to test.

STEP 4 | Test a request.

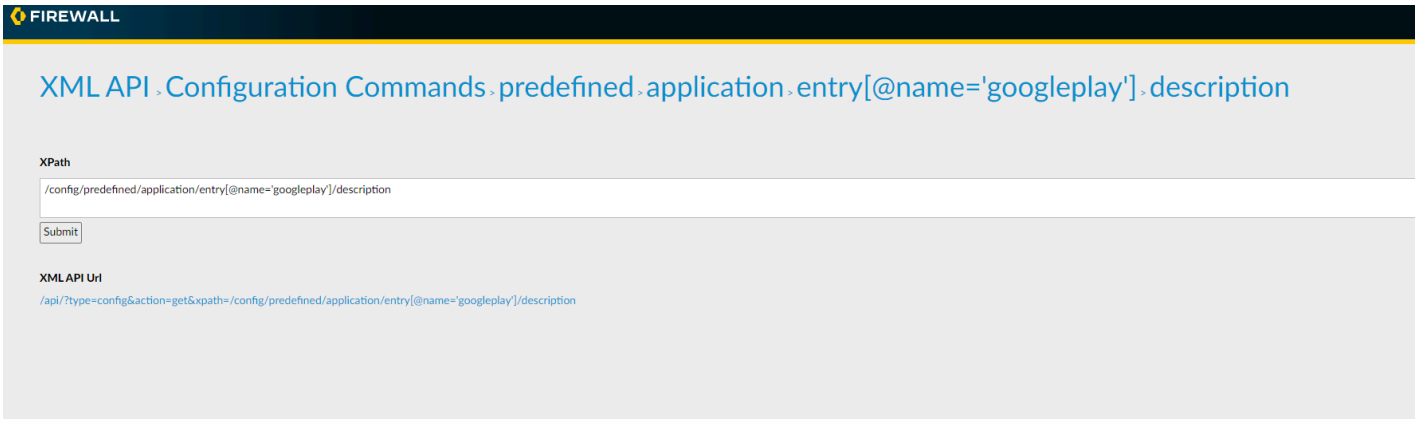
1. Select the URL to then test that request in the browser.



The browser shows the resulting XML response in the browser:



Along with the URL, the API browser also provides the XPath as necessary, as shown here for a description of a predefined application:



STEP 5 | When you're finished using the API browser, log out.

(PAN-OS version 11.0.2 and later versions of 11.0) Use the XML API logout button in the browser.

Use the CLI to Find XML API Syntax

Another method to determine the appropriate XML syntax and XPath for your API calls is through the command-line interface (CLI). This method works for **`type=op`** and **`type=config`** API calls.

To learn about changes to the latest version of CLI commands that affect corresponding PAN-OS XML API requests, see the PAN-OS CLI Quick Start.

Use the CLI to enable debug mode and then run the CLI command to receive the corresponding XML and XPath in the response.

STEP 1 | Access the CLI.

Use an SSH client or terminal to access your firewall or Panorama CLI.

STEP 2 | Enable debug mode.

Enter the following command:

```
debug cli on
```

STEP 3 | Run a CLI command.

Enter and run a CLI command. Example:

```
test url http://paloaltonetworks.com<request cmd="op"
 cookie="7581536015878829"
 uid="1206"><operations><test><url>http://paloaltonetworks.com</
url></test></operations></request>
```


STEP 4 | Use the resulting response to create an API call.

Use the **cmd** value and the XML elements within the **operations** tag to form the API call:

```
https://<firewall>/api/?type=op&cmd=<test><url>http://
paloaltonetworks.com</url></test>&key=<apikey>
```

*Depending on the CLI command, the XML tag values for **cmd** will vary. For example, here is a CLI command for showing firewall information: **run show system info***

The corresponding API call looks like this:

```
curl -X POST 'https://firewall/api?
type=op&cmd=<show><system><info></info></system></
show>&key=<apikey>"
```

Use the Web Interface to Find XML API Syntax

You can use the web interface along with the available debug console to explore the XML and XPath necessary for your API calls.

First log into the web interface and then open a separate window where you can view the corresponding XML and XPath.

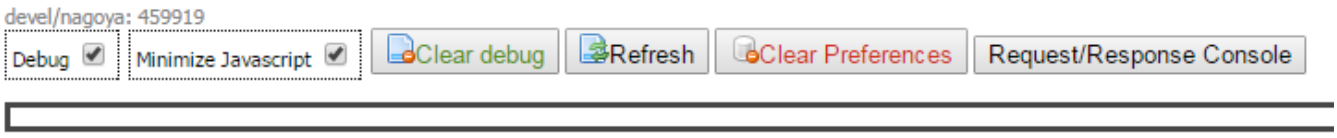
STEP 1 | Launch the web interface.

Launch a web browser and enter the firewall's IP address or hostname. Enter your user credentials.

STEP 2 | Launch the debug console.

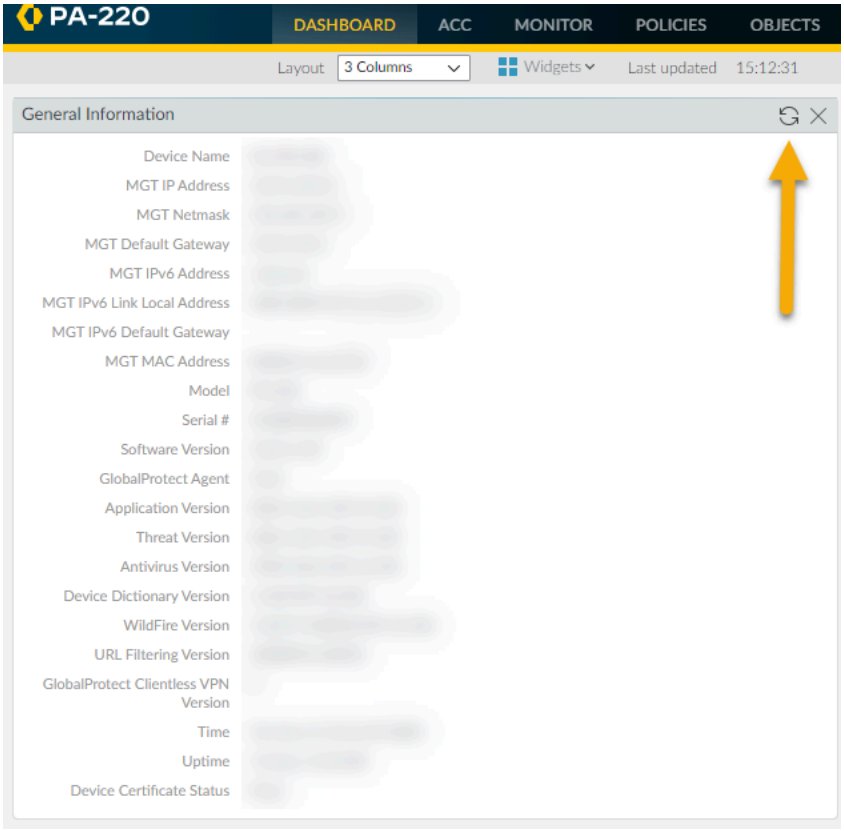
In a separate web browser window or tab, launch the debug console:

```
http://<firewall>/debug
```



STEP 3 | Perform the action you want to replicate through the API.

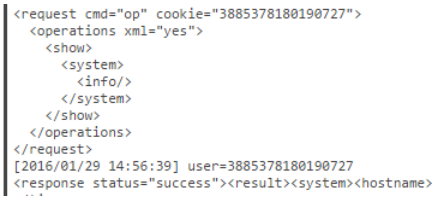
In the web browser, navigate to the menu and item or action that you want to perform.



To aid in finding the relevant XML, select **Clear** in the debug console just before you select the final menu or action.

STEP 4 | View the resulting XML syntax in the debug console.

In the debug console, select **Refresh** and then navigate through the console to the syntax related to your choice or action:



Example XML within debug console:

```
    <request cmd="op" cookie="3885378180190727">
<operations xml="yes">
  <show>
    <system>
      <info/>
    </system>
  </show>
</operations>
```



```
</request>
```

The corresponding API call looks like this:

```
curl -X POST 'https://firewall/api?
type=op&cmd=<show><system><info></info></system></
show>&key=<apikey>"
```


PAN-OS XML API Error Codes

The API response XML contains a status field and an error field. These are the available API error codes and names:

Error Code	Name	Description
400	Bad request	A required parameter is missing, an illegal parameter value is used.
403	Forbidden	Authentication or authorization errors including invalid key or insufficient admin access rights. Learn how to Get Your API Key.
1	Unknown command	The specific config or operational command is not recognized.
2-5	Internal errors	Check with technical support when seeing these errors.
6	Bad Xpath	The xpath specified in one or more attributes of the command is invalid. Check the API browser for proper xpath values.
7	Object not present	Object specified by the xpath is not present. For example, entry[@name='value'] where no object with name 'value' is present.
8	Object not unique	For commands that operate on a single object, the specified object is not unique.
10	Reference count not zero	Object cannot be deleted as there are other objects that refer to it. For example, address object still in use in policy.
11	Internal error	Check with technical support when seeing these errors.
12	Invalid object	Xpath or element values provided are not complete.
13	Object not found	Object presented in the request could not be found.
14	Operation not possible	Operation is allowed but not possible in this case. For example, moving a rule up one position when it is already at the top.

Error Code	Name	Description
15	Operation denied	Operation is allowed. For example, Admin not allowed to delete own account, Running a command that is not allowed on a passive device.
16	Unauthorized	The API role does not have access rights to run this query.
17	Invalid command	Invalid command or parameters.
18	Malformed command	The XML is malformed.
19-20	Success	Command completed successfully.
21	Internal error	Check with technical support when seeing these errors.
22	Session timed out	The session for this query timed out.

PAN-OS XML API Use Cases

The following use cases highlight the use of the PAN-OS XML API, either to reduce repetitive steps or to automate tasks normally you perform through the web interface or CLI.

Because the PAN-OS XML API uses a tree of XML nodes, in your API request you must specify the correct type and action along with the XPath Node Selection. See Explore the API to learn how to construct XML requests to be successful in using the API to meet your automation needs.

- Upgrade a Firewall to the Latest PAN-OS Version (API)
- Show and Manage GlobalProtect Users (API)
- Query a Firewall from Panorama (API)
- Upgrade PAN-OS on Multiple HA Firewalls through Panorama (API)
- Automatically Check for and Install Content Updates (API)
- Enforce Policy using External Dynamic Lists and AutoFocus Artifacts (API)
- Configure SAML 2.0 Authentication (API)
- Quarantine Compromised Devices (API)
- Manage Certificates (API)

Upgrade a Firewall to the Latest PAN-OS Version (API)

You can use the PAN-OS XML API to update your firewall with the latest PAN-OS and Content Release versions.

Because the PAN-OS XML API uses a tree of XML nodes, in your API request you must specify the correct type and action along with the XPath Node Selection. See Explore the API to learn how to construct XML requests.

STEP 1 | Download the latest content update.

Use the following request to first download the latest content update:

```
curl -X POST 'https://<firewall>/api?
type=op&cmd=<request><content><upgrade><download><latest/></
download></upgrade></content></request>'
```

If successful, the response contains a `jobid` that you can use to check on the status of your request.

```
     <response status="success" code="19">
  <result>
    <msg>
      <line>Download job enqueued with jobid 2</line>
    </msg>
    <job>2</job>
  </result>
</response>
```

STEP 2 | Check on the content download status.

Use the**jobid** to ensure that the content download completes successfully:

```
curl -X POST 'https://<firewall>/api?
type=op&cmd=<show><jobs><id>2</id></jobs></show>'
```

The response should include the following:

```
     <response status="success">…
```


STEP 3 | Install the latest content update.

Use the following request to install the newly downloaded content:

```
curl -X POST 'https://<firewall>/api?
type=op&cmd<request><content><upgrade><install> <version>latest</
version></install></upgrade></content></request>key=<apikey>'
```

If successful, the response contains a `jobid` that you can use to check on the status of your request.

```
     <response status="success" code="19">
  <result>
    <msg>
      <line>Content install job enqueued with jobid
       3</line>
    </msg>
    <job>3</job>
  </result>
</response>
```

STEP 4 | Check on the content installation status.

Use the `jobid` to ensure that the content installation completes successfully:

```
curl -X POST 'https://<firewall>/api?
type=op&cmd=<show><jobs><id>3</id></jobs></show>'
```

The response should include the following:

```
     <response status="success">…
```

STEP 5 | Check for the latest PAN-OS software update.

After installing the latest Content Release update, check for the latest available PAN-OS software updates:

```
curl -X POST 'https://<firewall>/api?
type=op&cmd=<request><system><software><check></check>
      </software></system></request>'
```

In the response, the first entry is the latest version of PAN-OS:

```
     <response
      status="success">
      <result>
```



```
        <sw-updates last-updated-at="2015/10/20 14:16:30">
        <msg />
        <versions>
        <version>7.1.0</version>
        <filename>PanOS_3000-7.1.0-c65</filename>
        <size>720</size>
        <size-kb>737504</size-kb>
        <released-on>2015/10/20 13:23:11</released-on>
        ...
```

STEP 6 | Download the latest PAN-OS software update.

1. In this case, the latest version is 7.1.0-c65, so download that version:

   ```
   curl -X POST 'https://<firewall>/api?
   type=op&cmd=<request><system><software><download><version>7.1.0-
   c65</version></download></software></system></request>'
   ```

2. Use the jobid in the response to ensure that the system update download completes successfully:

   ```
   curl -X POST 'https://<firewall>/api?
   type=op&cmd=<show><jobs><id>318</id></jobs></show>'
   ```

   The response should include the following:

   ```
          <response status="success">…
   ```

STEP 7 | Install the latest PAN-OS software update.

To install the latest system update, include the version in a software install request:

```
curl -X POST 'https://<firewall>/api?
type=op&cmd=<request><system><software><install><version>7.1.0-
c65</version></install></software></system></request>'
```

STEP 8 | Check on the software installation status.

Use the `jobid` in the response to ensure that the system update installs successfully:

```
curl -X POST 'https://<firewall>/api?
type=op&cmd=<show><jobs><id>320</id></jobs></show>'
```

The response should include the following:

```
      <response status="success">…
```


STEP 9 | Reboot the firewall.

After the system update installs successfully, trigger:

```
curl -X POST 'https://<firewall>/api?
type=op&cmd=<request><restart><system></system></restart></
request>'
```


Show and Manage GlobalProtect Users (API)

One common use of the PAN-OS XML API is to manage GlobalProtect users. You can use two API requests to view and then disconnect a Global Protect user who has been logged in for too long.

STEP 1 | View all GlobalProtect users.

Make a request to view all GlobalProtect users:

```
curl -X POST 'https://<firewall>/api?type=op&cmd=<show><global-
protect-gateway><current-user/>
       </global-protect-gateway></show>'
```

The response contains a list of users along with related information including IP addresses, logins, and client information:

```
     <response status="success">
  <result>
    <entry>
      <domain/>
      <islocal>yes</islocal>
      <username>dward</username>
      <computer>Dan's iPhone</computer>
      <client>Apple iOS 8.1.2</client>
      <vpn-type>Device Level VPN</vpn-type>
      <virtual-ip>192.168.2.1</virtual-ip>
      <public-ip>166.173.63.240</public-ip>
      <tunnel-type>SSL</tunnel-type>
      <login-time>Jan.22 01:50:36</login-time>
      <login-time-utc>1421916636</login-time-utc>
      <lifetime>2592000</lifetime>
    </entry>
  </result>
</response>
```

The `<login-time-utc>` field is the login date/time in UNIX time format (number of seconds elapsed since 00:00:00 1 Jan 1970). To find the list of users, filter the output for this field and compare the `<login-time-utc>` value to current date and time (or another date and time).

STEP 2 | Disconnect a GlobalProtect user.

Upon identifying the user that you want to disconnect, send a request that includes the GlobalProtect gateway, username, computer, and a **`force-logout`** reason:

```
curl -X POST 'https://<firewall>/api?type=op&cmd=<request><global-
protect-gateway><client-logout>
       <gateway>Home-N</gateway><user>dward</user><reason>force-
logout</reason>
       <computer>Dan's%20iPhone</computer></client-logout></global-
protect-gateway>
```



```
        </request>'
```

A successful response shows that the user has been successfully disconnected:

```
      <response status="success">
  <result>
    <response status="success">
      <gateway>Home-N</gateway>
      <domain>(null)</domain>
      <user>dward</user>
      <computer>Dan’s iPhone</computer>
    </response>
  </result>
</response>
```


Query a Firewall from Panorama (API)

The **target** parameter on Panorama allows you to redirect queries to a managed firewall. Redirecting queries to firewalls helps to reduce time and the number of steps required to issue repetitive commands. Use the scripting language of your choice to store firewall serial numbers and use them to issue a query to several firewalls.

Because the PAN-OS XML API uses a tree of XML nodes, in your API request you must specify the correct type and action along with the XPath Node Selection. See Explore the API to learn how to construct XML requests.

STEP 1 | Get a list of managed firewalls.

```
curl -X POST 'https://<panorama>/api?
type=op&cmd=<show><devices><all></all></devices></show>'
```

If you want to get a list of connected firewalls only, use

```
curl -X POST 'https://<panorama>/api?
type=op&cmd=<show><devices><connected></connected></
devices></show>'
```

The response includes the serial number (serial) of each firewall.

```
      <response

       status="success">
       <result>
       <devices>
       name="007200002517">
       <serial>007200002342</serial>
       <connected>yes</connected>
       <unsupported-version>no</unsupported-version>
       <deactivated>no</deactivated>
       <hostname>PM-6-1-VM</hostname>
       <ip-address>10.3.4.137</ip-address>
       <mac-addr />
       <uptime>81 days, 20:39:41</uptime>
       <family>vm</family>
       <model>PA-VM</model>
       <sw-version>6.1.3</sw-version>
       <app-version>555-3129</app-version>
       <av-version>2254-2693</av-version>
       <wildfire-version>91873-10.274</wildfire-version>
       <threat-version>555-3129</threat-version>
       <url-db>paloaltonetworks</url-db>
       <url-filtering-version>2016.02.02.416</url-filtering-
version>
       <logdb-version>6.1.3</logdb-version>
```



```
        <vpnclient-package-version />
        <global-protect-client-package-version>0.0.0</global-
protect-client-package-version>
        <vpn-disable-mode>no</vpn-disable-mode>
        <operational-mode>normal</operational-mode>
        <multi-vsys>no</multi-vsys>
        <vsys>
        name="vsys1">
        <display-name>vsys1</display-name>
        <shared-policy-status />
        <shared-policy-md5sum>4a0913667df83ff1098492e2e2ec1756</
shared-policy-md5sum>
        </entry>
        </vsys>
        </entry>
        <!--truncated -->
        </devices>
        </result>
        </response>
```

The response contains a `<serial>` XML element for each firewall.

STEP 2 | Collect firewall serial numbers.

In your script or code, store the firewall serial numbers returned in the response to the previous request.

STEP 3 | Query a firewall from Panorama.

A normal request to show system information on a firewall looks like this:

```
curl -X POST 'https://<firewall>/api?
type=op&cmd=<show><system><info></info></system></show>'
```

To directly target a firewall through Panorama, append the firewall serial number to the request:

```
curl -X POST 'https://<panorama>/api?
type=op&cmd=<show><system><info></info></system></
show>&target=<device-serial-number>'
```

A successful response should look like this:

```
     <response status="success">
  <result>
    <system>
      <hostname>firewall</hostname>
      <ip-address>10.41.0.8</ip-address>
      <netmask>255.255.224.0</netmask>
      <default-gateway>10.41.0.1</default-gateway>
      <is-dhcp>no</is-dhcp>
      <ipv6-address>unknown</ipv6-address>
```



```
      <ipv6-link-local-address>fe80::21c:17cf:feff:c04a/64</ipv6-
link-local-address>
      <ipv6-default-gateway/>
      <mac-address>00:1b:17:fc:c0:4a</mac-address>
      <time>Tue Oct 27 13:39:09 2015</time>
      <uptime>12 days, 0:05:26</uptime>
      <devicename>pm-firewall</devicename>
      <family>3000</family>
      <model>PA-3020</model>
      <serial>001802000104</serial>
      <sw-version>7.1.0-c54</sw-version>
      <global-protect-client-package-version>2.0.0</global-protect-
client-package-version>
      <app-version>537-2965</app-version>
      <app-release-date>2015/10/26 18:10:48</app-release-date>
      <av-version>2149-2586</av-version>
      <av-release-date>2015/10/26 15:31:55</av-release-date>
      <threat-version>537-2965</threat-version>
      <threat-release-date>2015/10/26 18:10:48</threat-release-
date>
      <wf-private-version>0</wf-private-version>
      <wf-private-release-date>unknown</wf-private-release-date>
      <url-db>paloaltonetworks</url-db>
      <wildfire-version>80683-89773</wildfire-version>
      <wildfire-release-date>unknown</wildfire-release-date>
      <url-filtering-version>2015.10.27.226</url-filtering-version>
      <global-protect-datafile-version>1445974904</global-protect-
datafile-version>
      <global-protect-datafile-release-date>2015/10/27 19:41:44</
global-protect-datafile-release-date>
      <logdb-version>7.0.9</logdb-version>
      <platform-family>3000</platform-family>
      <vpn-disable-mode>off</vpn-disable-mode>
      <multi-vsys>on</multi-vsys>
      <operational-mode>normal</operational-mode>
    </system>
  </result>
</response>
```

Repeat this request for each managed or connected firewall.

Upgrade PAN-OS on Multiple HA Firewalls through Panorama (API)

This use case highlights the ability of the PAN-OS XML API to automate a more complex procedure, namely upgrading firewalls set up as active-passive high-availability (HA) pair. Normally, this procedure involves multiple, manual steps on individual firewalls.

This is a high-level overview of the steps you must take in this procedure. You script or application must incorporate error-checking and logic to implement this sequence of steps.

Because the PAN-OS XML API uses a tree of XML nodes, in your API request you must specify the correct type and action along with the XPath Node Selection*. See* Explore the API *to learn how to construct XML requests.*

STEP 1 | Check for the latest PAN-OS software update through Panorama

Check for the latest available PAN-OS software updates. Include the firewall serial number in your request:

```
curl -X POST 'https://panorama/api?
type=op&cmd=<request><system><software><check></check></software></
system></request>&target=007200002517&key=<apikey>"
```

The response contains an array of results sorted to show the latest version first:

```
<response status="success">
  <result>
    <sw-updates last-updated-at="2016/02/03 08:29:09">
      <msg />
      <versions>
        >
          <version>7.1</version>
          <filename>PanOS_vm-7.1</filename>
          <size>540</size>
          <size-kb>553964</size-kb>
          <released-on>2016/02/02 10:57:20</released-on>
          <release-notes><![CDATA[https://10.44.2.19/
updates/ReleaseNotes.aspx?type=sw&versionNumber=7.1.0-
c158&product=panos&platform=vm]]></release-notes>
          <downloaded>no</downloaded>
          <current>no</current>
          <latest>yes</latest>
        </entry>
<!-- truncated -->
      </versions>
    </sw-updates>
  </result>
</response>
```


STEP 2 | Download the latest PAN-OS software update.

1. In this case, the latest version is 7.1.0-c65, so download that version:

```
curl -X GET
'https://<firewall>/api/?
type=op&cmd=<request><system><software><download><version>7.1.0
-c65</version></download></software></system></request>'
```

2. Use the `jobid` in the response to ensure that the system update download completes successfully:

```
curl -X POST 'https://<firewall>/api?type=op&action=get&job-
id=318'
```

The response should include the following:

```
<response status="success">…
```

STEP 3 | Install the latest PAN-OS software update.

To install the latest system update, include the version in a software install request:

```
curl -X POST 'https://<firewall>/api?
type=op&cmd=<request><system><software><install><version>7.1.0-
c65</version></install></software></system></request>'
```

STEP 4 | Check on the software installation status.

Use the `jobid` in the response to ensure that the system update installs successfully:

```
curl -X POST 'https://<firewall>/api?type=op&action=get&job-
id=<jobid>'
```

The response should include the following:

```
<response status="success">…
```

STEP 5 | Get a list of connected firewalls.

Get a list of connected firewalls that Panorama manages:

```
curl -X POST 'https://panorama/api?
type=op&cmd=<show><devices><https://<panorama>/api/?
type=op&cmd=<show><devices><connected></connected></devices></
show>"
```

The response includes the serial number (`serial`) of each firewall.

```
<response status="success">
:  <result>
:    <devices>
:       name="007200002517">
:        <serial>007200002342</serial>
```



```
:         <connected>yes</connected>
:         <unsupported-version>no</unsupported-version>
:         <deactivated>no</deactivated>
:         <hostname>PM-6-1-VM</hostname>
:         <ip-address>10.3.4.137</ip-address>
:         <mac-addr />
:         <uptime>81 days, 20:39:41</uptime>
        <family>vm</family>
        <model>PA-VM</model>
        <sw-version>6.1.3</sw-version>
        <app-version>555-3129</app-version>
        <av-version>2254-2693</av-version>
        <wildfire-version>91873-10.274</wildfire-version>
        <threat-version>555-3129</threat-version>
        <url-db>paloaltonetworks</url-db>
        <url-filtering-version>2016.02.02.416</url-filtering-
version>
        <logdb-version>6.1.3</logdb-version>
        <vpnclient-package-version />
        <global-protect-client-package-version>0.0.0</global-
protect-client-package-version>
        <vpn-disable-mode>no</vpn-disable-mode>
        <operational-mode>normal</operational-mode>
        <multi-vsys>no</multi-vsys>
        <vsys>
           name="vsys1">
            <display-name>vsys1</display-name>
            <shared-policy-status />
            <shared-policy-
md5sum>4a0913667df83ff1098492e2e2ec1756</shared-policy-md5sum>
          </entry>
        </vsys>
      </entry>

   <!--truncated -->

    </devices>
  </result>
</response>
```

The response contains a <serial> XML element that contains each firewall serial number.

STEP 6 | Check for the latest PAN-OS software update.

Check to see if new software is available on your HA pair:

```
curl -X POST 'https://panorama/api?
type=op&cmd=<request><system><software><check></check></software></
system></request>&target=<serialnumber>&key=<apikey>"
```

The response contains an array of results sorted to show the latest version first:

```
<response status="success">
<result>
<sw-updates last-updated-at="2016/02/03 08:29:09">
<msg />
```



```
<versions>
<version>7.1</version>
<filename>PanOS_vm-7.1</filename>
<size>540</size>
<size-kb>553964</size-kb>
<released-on>2016/02/02 10:57:20</released-on>
<release-notes><![CDATA[https://10.44.2.19/updates/
ReleaseNotes.aspx?type=sw&versionNumber=7.1.0-
c158&product=panos&platform=vm]]></release-notes>
<downloaded>no</downloaded>
<current>no</current>
<latest>yes</latest>
</entry>
<!-- truncated -->
</versions>
</sw-updates>
</result>
</response>
```

STEP 7 | Download the latest PAN-OS software update.

After determining the latest system update, download it to both firewalls in the HA pair:

```
curl -X POST 'https://panorama/api?
type=op&cmd=<request><system><software><download><version>7.1</
version></download></software></system></
request>&target=<serialnumber>&key=<apikey>"
```

The response contains a job ID:

```
<response status="success" code="19">
  <result>
    <msg>
      <line>Download job enqueued with jobid 3448</line>
    </msg>
    <job>3448</job>
  </result>
</response>
```

Use the job ID to check on the download status:

```
curl -X POST 'https://panorama/api?
type=op&cmd=<show><jobs><id>3448</id></jobs></
show>&target=<serialnumber>&key=<apikey>"
```

The response contains a job status of FIN when the download is complete:

```
<response status="success">
  <result>
    <job>
      <tenq>2016/02/03 08:32:00</tenq>
      <id>3448</id>
      <user/>
      <type>Downld</type>
      <status>FIN</status>
```



```
      <stoppable>no</stoppable>
      <result>OK</result>
      <tfin>08:32:10</tfin>
      <progress>08:32:10</progress>
      <details>
        <line>Successfully downloaded</line>
        <line>Preloading into software manager</line>
        <line>Successfully loaded into software manager</line>
      </details>
      <warnings/>
    </job>
  </result>
</response>
```

STEP 8 | Suspend the active HA firewall.

Suspend the active firewall in your high-availability firewall pair:

```
curl -X POST 'https://panorama/api?type=op&cmd=<request><high-
availability><state><suspend></suspend></state></high-
availability></request>&target=<serialnumber>&key=<apikey>"
```

The response confirms the active firewall has been suspended:

```
<response status="success">
  <result>Successfully changed HA state to suspended</result>
</response>
```

STEP 9 | Install the latest software update on the suspended HA pair.

After suspending the active HA firewall, install the system update on it:

```
curl -X POST 'https://panorama/api?
type=op&cmd=<request><system><software><install><version>version</
version></install></software></system></
request>&target=<serialnumber>&key=<apikey>"
```

The response shows the system update is queued:

```
<response status="success" code="19">
  <result>
    <msg>
      <line>Software install job enqueued with jobid 3453. Run
 'show jobs id 3453' to monitor its status. Please reboot the
 device after the installation is done.</line>
    </msg>
    <job>3453</job>
  </result>
</response>
```


STEP 10 | Check on the software installation status.

Use the `jobid` in the response to ensure that the system update installs successfully:

```
curl -X POST 'https://<panorama>/api?type=op&action=get&job-
id=jobid&target=<serialnumber>&key=<apikey>
```

The response should include the following:

```
<response status="success">…
```

STEP 11 | Reboot the suspended HA peer.

After installing the latest system update, reboot the suspended HA peer:

```
curl -X POST 'https://panorama/api?
type=op&cmd=<request><restart><system></system></restart></
request>&target=<serialnumber>&key=<apikey>"
```

STEP 12 | Verify that the upgrade is successful.

Show system information on your upgraded HA peer to ensure it has the latest system update and is operational:

```
curl -X POST 'https://panorama/api?
type=op&cmd=<show><system><info></info></system></
show>&target=<serialnumber>&key=<apikey>"
```

STEP 13 | Makes the suspended HA peer active.

After you verify that the system update on the suspended HA peer is successful, make it active again:

```
curl -X POST 'https://panorama/api?type=op&cmd=<request><high-
availability><state><functional></functional></state></high-
availability></request>&target=<serialnumber>&key=<apikey>"
```

The response confirms the active firewall is now active:

```
 <response status="success">
  <result>Successfully changed HA state to functional</result>
</response>
```

STEP 14 | Install the system update on the passive HA peer.

Once the suspended HA firewall is active, you can then repeat steps 5-8 on the now passive HA peer.

Automatically Check for and Install Content Updates (API)

Using the XML API, you can programmatically check and install new content updates, including antivirus, WildFire, and GlobalProtect updates. Check for new updates available and download updates that have been released for at least one week.

Download, upgrade, and installation requests are asynchronous. The API responds with a job ID while it processes your request. In your subsequent request, you use this job ID to check on the result of your original request:

```
curl -X POST 'https://firewall/api?type=op&cmd=<show><jobs><id></
id></jobs></show>&key=<apikey>"
```

STEP 1 | Check for installed content on your firewall. Run the following request to view current system information:

```
curl -X POST 'https://firewall/api?
type=op&cmd=<show><system><info></info></system></
show>&key=<apikey>"
```

STEP 2 | Confirm that the API response to the request in the previous step includes the currently installed updates on your firewall:

```
<response status="success">
  <result>
    <system>
      <hostname>pm-firewall</hostname>
      <ip-address>10.47.0.8</ip-address>
      <netmask>255.255.254.0</netmask>
      <default-gateway>10.47.0.1</default-gateway>
      <is-dhcp>no</is-dhcp>
      <ipv6-address>unknown</ipv6-address>
      <ipv6-link-local-address>fe80::21b:17ff:feff:c04a/64</ipv6-
link-local-address>
      <ipv6-default-gateway/>
      <mac-address>00:1b:17:ff:c0:4a</mac-address>
      <time>Mon Jul 11 17:51:37 2016</time>
      <uptime>11 days, 7:38:34</uptime>
      <devicename>pm-firewall</devicename>
      <family>3000</family>
      <model>PA-3020</model>
      <serial>0018010.2104</serial>
      <sw-version>7.1.3</sw-version>
      <global-protect-client-package-version>2.0.0</global-protect-
client-package-version>
      <app-version>598-3427</app-version>
      <app-release-date>2016/07/09 22:30:55</app-release-date>
      <av-version>2416-2855</av-version>
      <av-release-date>2016/07/10 11:27:57</av-release-date>
      <threat-version>598-3427</threat-version>
```



```
      <threat-release-date>2016/07/09 22:30:55</threat-release-
date>
      <wf-private-version>0</wf-private-version>
      <wf-private-release-date>unknown</wf-private-release-date>
      <url-db>paloaltonetworks</url-db>
      <wildfire-version>80426-81466</wildfire-version>
      <wildfire-release-date>2016/07/11 17:45:11</wildfire-release-
date>
      <url-filtering-version>2016.07.11.248</url-filtering-version>
      <global-protect-datafile-version>1468280405</global-protect-
datafile-version>
      <global-protect-datafile-release-date>2016/07/11 23:40:05</
global-protect-datafile-release-date>
      <logdb-version>7.0.9</logdb-version>
      <platform-family>3000</platform-family>
      <vpn-disable-mode>off</vpn-disable-mode>
      <multi-vsys>on</multi-vsys>
      <operational-mode>normal</operational-mode>
    </system>
  </result>
</response>
```

STEP 3 | Note the currently installed versions for the following updates, so that you can compare the values after you check for the latest updates:

- global-protect-client-package-version: GlobaProtect
- app-version: Application and threat signatures.
- av-version: Antivirus signatures
- wildfire-version: WildFire malware and antivirus signatures

STEP 4 | Check for new, available updates with the following requests and store the version field in the response, which is the `version` field for GlobalProtect, and the `app-version` field for all others:

- GlobalProtect:

```
curl -X POST 'https://firewall/api?type=op&cmd=<request><global-
protect-client><software><check></check></software></global-
protect-client></request>&key=<apikey>"
```

- WildFire:

```
curl -X POST 'https://firewall/api?
type=op&cmd=<request><wildfire><upgrade><check></check></upgrade></
wildfire></request>&key=<apikey>"
```

- Application & Threat:

```
curl -X POST 'https://firewall/api?
type=op&cmd=<request><content><upgrade><check></check></upgrade></
content></request>&key=<apikey>"
```

- Antivirus:

```
curl -X POST 'https://firewall/api?type=op&cmd=<request><anti-
virus><upgrade><check></check></upgrade></anti-virus></
request>&key=<apikey>"
```

Example response:

```
<response status="success">
<result>
<sw-updates last-updated-at="2016/05/19 14:34:34">
<msg/>
<versions>
<entry>
<version>4.0.0-c16</version>
<filename>PanGP-4.0.0-c16</filename>
<size>44</size>
<size-kb>45321</size-kb>
<released-on>2016/07/08 15:41:18</released-on>
<release-notes>
<![CDATA[
https://firewall/updates/ReleaseNotes.aspx?
type=sw&versionNumber=4.0.0-c16&product=gpclient&platform=any
]]>
</release-notes>
<downloaded>no</downloaded>
<current>no</current>
<latest>no</latest>
<uploaded>no</uploaded>
</entry>
```



```
<!--TRUNCATED-->
```

Take note of the `released-on` XML field to verify that updates have been released for at least a week.

STEP 5 | In your script or code, compare the version values for currently installed updates to new, available updates. It is recommended that you only install updates that have been available for at least a week.

STEP 6 | Download the latest content updates with these requests:

- GlobalProtect:

```
curl -X POST 'https://firewall/api?type=op&cmd=<request><global-
protect-client><software><download><version>versionnumber</
version></download></software></global-protect-client></
request>&key=<apikey>"
```

- WildFire:

```
curl -X POST 'https://firewall/api?
type=op&cmd=<request><wildfire><upgrade><download><latest></
latest></download></upgrade></wildfire></request>&key=<apikey>"
```

- Application & Threat:

```
curl -X POST 'https://firewall/api?
type=op&cmd=<request><content><upgrade><download><latest></
latest></download></upgrade></content></request>"
```

- Antivirus:

```
curl -X POST 'https://firewall/api?type=op&cmd=<request><anti-
virus><upgrade><download><latest></latest></download></upgrade></
anti-virus></request>&key=<apikey>"
```

The response contains a job ID that you can use to check on the status of the request. Example:

```
<response status="success" code="19">
  <result>
    <msg>
      <line>Content install job enqueued with jobid 299</line>
    </msg>
    <job>299</job>
  </result>
</response>
```

Learn more about Asynchronous and Synchronous Requests to the PAN-OS XML API.

STEP 7 | Install the latest content updates with these requests:

- GlobalProtect:

```
curl -X POST 'https://firewall/api?type=op&cmd=<request><global-
protect-client><software><activate><version>versionnumber</
version></activate></software></global-protect-client></
request>&key=<apikey>"
```

- WildFire:

```
curl -X POST 'https://firewall/api?
type=op&cmd=<request><wildfire><upgrade><install><version>latest</
version></install></upgrade></wildfire></request>&key=<apikey>"
```

- Application & Threat:

```
curl -X POST 'https://firewall/api?
type=op&cmd=<request><content><upgrade><install>latest</latest></
install></upgrade></content></request>&key=<apikey>"
```

- Antivirus:

```
curl -X POST 'https://firewall/api?type=op&cmd=<request><anti-
virus><upgrade><install><version>latest</version></install></
upgrade></anti-virus></request>&key=<apikey>"
```

The response contains a job ID that you can use to check on the status of the request.

Enforce Policy using External Dynamic Lists and AutoFocus Artifacts (API)

This use case allows you to use data from AutoFocus threat intelligence to create an external dynamic list for your firewall.

Use the AutoFocus API to export AutoFocus artifacts (IP addresses, domains, or URLs) as an export list that you can host on a web server. Learn more about AutoFocus in AutoFocus documentation. Then use the PAN-OS XML API to add this URL as an external dynamic list to enforce policy dynamically on the firewall. Learn more about how to use an external dynamic list in policy.

To use AutoFocus, you must first register and activate AutoFocus.

STEP 1 | Build an AutoFocus export list. For example, if you want to block potential attacks from the Sofacy group, search for Sofacy as the Tag, and then add the appropriate artifacts shown within the File Analysis tab, such as DNS Activity, HTTP Requests, and Connection Activity.

```
Use the AutoFocus API to export the AutoFocus artifacts. Include
 you the AutoFocus API key, the label of the export list, and
 specify that the list should be formatted for a PAN-OS block list.
 ("panosFormatted":true):
       curl -X POST -H "Content-Type: application/json" -d '{
    "apiKey": "<apikey>",
    "label": "<export-list-name>",
    "panosFormatted": true
}' "https://autofocus.paloaltonetworks.com/api/v1.0/export"
```

The response contains a list of IP addresses, domains, or URLs, depending on the artifacts you save:

```
      {
    "bucket_info": {
        "daily_points": 10.2,
        "minute_points": 200
    },
    "export_list": [
        "176.31.112.10",
        "31.220.43.99",
        "40.76.58.209",
        "62.113.232.196",
        "95.215.47.207"
    ]
}
```

STEP 2 | Host the export list as a text file on an external web server. To ensure that you have the latest list of artifacts, frequently refresh the hosted list.

STEP 3 | Add the URL for the export list to an external dynamic list. In this example the external dynamic list uses IP addresses:

```
curl -X POST 'https://firewall/api?type=config&action=set&xpath=/
config/devices/entry[@name='localhost.localdomain']/vsys/
entry[@name='vsys1']/external-list/entry[@name='export-list-name']/
type/ip&element=<url><edl-list-url></url><recurring><five-minute/
></recurring>&key=<apikey>"
```

STEP 4 | Add the external dynamic list as match criteria in a security policy rule. In this example, the rule denies access to IP addresses on the external dynamic list for all users on your network:

```
curl -X POST 'https://firewall/api?type=config&action=set&xpath=/
config/devices/entry[@name='localhost.localdomain']/
vsys/entry[@name='vsys1']/rulebase/security/rules/
entry[@name='<security-policy-rulename>']@element=<to><member>any</
member></to><from><member>any</member></from><source>any</
source><destination><member><edl-list-name></member></
destination><source-user><member>any</member></source-
user><service><member>application-default</member></
service><action>deny</action>&key=<apikey>"
```

STEP 5 | Commit the changes to the firewall:

```
curl -X POST 'https://firewall/api?type=commit&cmd=<commit></
commit>&key=<apikey>"
```

You must commit only once when you add the reference to the EDL in a policy rule. Any changes to the external dynamic list do not require a commit.

Configure SAML 2.0 Authentication (API)

Use the PAN-OS XML API to automate the configuration of SAML 2.0 single sign-on (SSO) and single logout (SLO). To configure SAML using the API, create scripts that import the SAML metadata file, create a SAML authentication profile, add users and user groups, and assign the authentication profile to firewall services. The following workflow provides an example of how to configure SAML using the XML API.

Because the PAN-OS XML API uses a tree of XML nodes, in your API request you must specify the correct type and action along with the XPath Node Selection. See Explore the API to learn how to construct XML requests.

STEP 1 | (Recommended) **Import a metadata file from the IdP**

The metadata file contains registration information and the certificate that the IdP uses to sign SAML messages. If you import a metadata file, you do not need to independently Create a SAML Identity Provider (IdP) server profile. Include the metadata file path and SAML server profile name in your GET request:

- **key**: API key
- **file**: file path to SAML metadata file. The metadata file contains registration information, as well as the certificate that the IdP uses to sign SAML messages. Export the metadata file from the IdP to a client system that the firewall can access. The certificate specified in the file must meet SAML requirements. Refer also to your IdP documentation for instructions.
- **profile-name**: passphrase, up to 31 characters

```
curl -k -F file=@filename.txt -g 'https://<firewall>/api/?
type=import&category=idp-metadata&profile-name=<profilename>'
```

If you perform this step, you can skip Step 2, Create a SAML Identity Provider (IdP) server profile.

STEP 2 | Create a SAML Identity Provider (IdP) server profile

If you do not import a metadata file, include IdP configuration parameters in your GET request to create a SAML IdP server profile:

- **key**: API key
- **vsys**: location, example values: shared, vsys1, vsys2
- **name**: server profile name
- **entity-id**: identity provider id
- **certificate**: **(**Best Practice**)** identity provider certificate
- **sso-url**: identity provider SSO URL
- **slo-url:** identity provider SLO URL
- **sso-binding:** SSO SAML HTTP binding, acceptable values: post, redirect
- **ssl-binding:** SSL SAML HTTP binding, acceptable values: post, redirect
- **max-clock-skew**: difference in system time as measured in seconds between firewall and IdP. The default value is 60 with a range of 1-900.
- validate-idp-certificate: **(**Best Practice**)** specify whether you want to validate the IdP certificate. The default value is yes.
- **want-auth-requests-signed**: specify whether the IdP expects a digital signature on authentication requests. The default value is no.

```
curl -X POST 'https://firewall/api?type=config&action=set&xpath=/
config/shared/server-profile/saml-idp/entry[@name='<server-profile-
name>']&element=<certificate><cert-name></certificate><entity-
id><https://example.com/sso></entity-id><sso-url><https://
example.com/sso></sso-url><sso-bindings><post></sso-
bindings><slo-url><https://example.com/slo></slo-url><slo-
bindings>post</slo-bindings><max-clock-skew><max-clock-skew></
max-clock-skew><validate-idp-certificate><yes></validate-idp-
certificate><want-auth-requests-signed><yes></want-auth-requests-
signed>"
```


STEP 3 | Create a SAML authentication profile using the PAN-OS XML API

Include SAML authentication profile parameters in your GET request:

- **key**: API key
- **authentication-profile:** authentication profile name
- **enable-single-logout:** specify whether you want to enable SAML single logout. The default value is no.
- **request-signing-certificate:** request signing certificate name
- **server-profile:** SAML Identity Provider (IdP) server profile name
- **certificate-profile:** certificate profile name
- **attribute-name-username:** SAML username attribute
- **attribute-name-usergroup:** SAML user group attribute
- **attribute-name-access-domain:** SAML admin domain attribute
- **attribute-name-admin-role:** SAML admin role attribute

```
curl -X POST 'https://firewall/api?type=config&action=set&xpath=/
config/shared/authentication-profile/entry[@name='<authentication-
profile-name>']/method/saml-idp&element=<enable-single-logout>no</
enable-single-logout><request-signing-certificate><certificate-
name></request-signing-certificate><server-profile><server-
profile-name></server-profile><certificate-profile>profile-name</
certificate-profile><attribute-name-username><username></attribute-
name-username><attribute-name-usergroup><usergroup></attribute-
name-usergroup><attribute-name-access-domain><access-domain></
attribute-name-access-domain><attribute-name-admin-role><admin-
role></attribute-name-admin-role>"
```

STEP 4 | Add users and user groups that are allowed to authenticate with this authentication profile

Include profile name and member list in your request:

- **key**: API key
- **authentication-profile:** authentication profile name
- **member**: users or user groups. To include specific users or groups, include them in brackets: `[member1,member 3]`. To include all users, include**all**.

```
curl -X POST 'https://firewall/api?type=config&action=set&xpath=/
config/shared/authentication-profile/entry[@name='<authentication-
profile-name>']/allow-list&element=<member><all></member>"
```


STEP 5 | Assign the authentication profile to firewall services that require authentication

For example, to assign the authentication profile to a superuser administrator account for web access, include these parameters in your GET request:

- **key**: API key
- **name**: admin username
- **authentication-profile**: name of the SAML authentication profile

```
curl -X POST 'https://firewall/api?type=config&action=set&xpath=/
config/mgt-config/users/
entry[@name='<adminname>']&element=<permissions><role-
based><superuser>yes</superuser></role-based></
permissions><authentication-profile><authprofilename></
authentication-profile>"
```


Quarantine Compromised Devices (API)

You can use this XML API to identify compromised devices by adding them to a quarantine list, which you can then use to block GlobalProtect users from connecting those devices to a gateway. See the following for examples of XML API requests to manage device quarantine.

- Add a Device to a Quarantine List
- List Quarantined Devices
- Delete a Device From the Quarantine List

Add a Device to a Quarantine List

You can use the XML API to write one or more compromised devices to the quarantine list on the firewall.

The following example shows the basic syntax of the XML API request path to add one or more identified devices to the quarantine list on the firewall.

```
    https://<firewall>/api/?type=op&cmd=<set><quarantine><data></
data></quarantine></set>
```

The <data></data> content identifies the device(s) to be quarantined and the following table lists the tags in the content. All tags listed in the table are required.

Tag	Description	Notes
<iot-message>	Denotes message	
<version>	XML version	Content is "1.0"
<type>	Indicates a request to update	Content is “update”
<vsys>	vsys of the firewall that has the quarantine list	Content is the vsys
<payload>	Denotes payload	
<quarantine-add>	Identifies action to add quarantine device	
<entry hostid="">	Attribute hostid is the compromised device ID. Can have multiple entries. Include one entry per device to be quarantined	Attribute hostid is required

Tag	Description	Notes
<serialno>	Serial number of the device to be quarantined	Content optional
<reason>	Reason for quarantine	Content required. No spaces allowed in content
<source>	Source device or application from which this quarantine device was added to the quarantine list	Content required
<quarantine-ts>	Quarantine timestamp. Time when device was added to quarantine list.	Content required

The following is an example of <data></data> content to add one device to the quarantine list.

```
<iot-message><version>1.0</version><type>update</type><vsys>vsys1</
vsys><payload><quarantine-add><entry
hostid="host3"><serialno>serial1</serialno><reason>admin1</
reason><source>litest1</source><quarantine-ts>1234</quarantine-ts></
entry></quarantine-add></payload></iot-message>
```

The following is an example of the <data></data> content to add more than one device to the quarantine list.

```
<iot-message><version>1.0</version><type>update</
type><vsys>vsys1</vsys><payload><quarantine-add><entry
 hostid="host9"><serialno>123uabcd2</serialno><reason>Magnifier</
reason><source>Magnifier</source><quarantine-ts>7890</quarantine-
ts>&</entry><entry hostid="host8"><serialno>309ufwi88</
serialno><reason>Cortex</reason><source>Cortex</source><quarantine-
ts>4567</quarantine-ts></entry></quarantine-add></payload></iot-
message
```

STEP 1 | Encode the data content.

You must encode the data content twice before you submit it with the XML API request. First HTML encode the content and then URL encode the HTML encoded content.

1. HTML encode the content.

   One approach is to use the CLI. See Use the CLI to Find XML API Syntax for details about how to use the CLI to determine the syntax of this request. The following command outputs an HTML encoded string.

   ```
   set quarantine data '<iot-message><version>1.0</
   version><type>update</type><vsys>vsys1</
   vsys><payload><quarantine-add><entry
   ```



```
 hostid="host3"><serialno>serial1</serialno><reason>admin1</
reason><source>test1</source><quarantine-ts>1234</quarantine-
ts></entry></quarantine-add></payload></iot-message>'
```

The output from this command includes the following. The <data></data> content is the HTML encoded version of your original data content.

```
<request cmd="op" cookie="3515166656333795"
 uid="10.2"><operations><set><quarantine><data>&lt;iot-
message&gt;&lt;version&gt;1.0&lt;/
version&gt;&lt;type&gt;update&lt;/
type&gt;&lt;vsys&gt;vsys1&lt;/
vsys&gt;&lt;payload&gt;&lt;quarantine-add&gt;&lt;entry
 hostid=&quot;host3&quot;&gt;&gt;&lt;serialno&gt;serial1&lt;/
serialno&gt;&lt;reason&gt;admin1&lt;/
reason&gt;&lt;source&gt;test1&lt;/source&gt;&lt;quarantine-
ts&gt;1234&lt;/quarantine-ts&gt;&lt;/entry&gt;&lt;/quarantine-
add&gt;&lt;/payload&gt;&lt;/iot-message&gt;</data></
quarantine></set></operations></request>
```

2. URL encode the HTML encoded data content.

   The example below shows the data content that we HTML encoded and now want to URL encode.

```
&lt;iot-message&gt;&lt;version&gt;1.0&lt;/
version&gt;&lt;type&gt;update&lt;/
type&gt;&lt;vsys&gt;vsys1&lt;/
vsys&gt;&lt;payload&gt;&lt;quarantine-add&gt;&lt;entry
 hostid=&quot;host3&quot;&gt;&lt;serialno&gt;serial1&lt;/
serialno&gt;&lt;reason&gt;admin1&lt;/
reason&gt;&lt;source&gt;test1&lt;/source&gt;&lt;quarantine-
ts&gt;1234&lt;/quarantine-ts&gt;&lt;/entry&gt;&lt;/quarantine-
add&gt;&lt;/payload&gt;&lt;/iot-message&gt;
```

   The URL encoded content for the data content shown above is as follows.

```
%26lt%3Biot-message%26gt%3B%26lt%3Bversion%26gt%3B1.0%26lt%3B
%2Fversion%26gt%3B%26lt%3Btype%26gt%3Bupdate%26lt%3B%2Ftype
%26gt%3B%26lt%3Bvsys%26gt%3Bvsys1%26lt%3B%2Fvsys%26gt%3B%26lt
%3Bpayload%26gt%3B%26lt%3Bquarantine-add%26gt%3B%26lt%3Bentry
%20hostid%3D%26quot%3Bhost3%26quot%3B%26gt%3B%26gt%3B%26lt
%3Bserialno%26gt%3Bserial1%26lt%3B%2Fserialno%26gt%3B%26lt
%3Breason%26gt%3Badmin1%26lt%3B%2Freason%26gt%3B%26lt%3Bsource
%26gt%3Btest1%26lt%3B%2Fsource%26gt%3B%26lt%3Bquarantine-ts
%26gt%3B1234%26lt%3B%2Fquarantine-ts%26gt%3B%26lt%3B%2Fentry
%26gt%3B%26lt%3B%2Fquarantine-add%26gt%3B%26lt%3B%2Fpayload
%26gt%3B%26lt%3B%2Fiot-message%26gt%3B
```


STEP 2 | Make a request to add your device to the quarantine list on the firewall.

The following example shows a cURL request to add a single compromised device to the quarantine list.

```
curl -X POST 'https://<firewall>/api?key=<api
 key>&type=op&cmd=<set><quarantine><data>%26lt%3Biot-message%26gt
%3B%26lt%3Bversion%26gt%3B1.0%26lt%3B%2Fversion%26gt%3B%26lt%3Btype
%26gt%3Bupdate%26lt%3B%2Ftype%26gt%3B%26lt%3Bvsys%26gt%3Bvsys1%26lt
%3B%2Fvsys%26gt%3B%26lt%3Bpayload%26gt%3B%26lt%3Bquarantine-add
%26gt%3B%26lt%3Bentry%20hostid%3D%26quot%3Bhost3%26quot%3B%26gt
%3B%26lt%3Bserialno%26gt%3Bserial1%26lt%3B%2Fserialno%26gt%3B%26lt
%3Breason%26gt%3Badmin1%26lt%3B%2Freason%26gt%3B%26lt%3Bsource
%26gt%3Btest1%26lt%3B%2Fsource%26gt%3B%26lt%3Bquarantine-ts%26gt
%3B1234%26lt%3B%2Fquarantine-ts%26gt%3B%26lt%3B%2Fentry%26gt%3B
%26lt%3B%2Fquarantine-add%26gt%3B%26lt%3B%2Fpayload%26gt%3B%26lt%3B
%2Fiot-message%26gt%3B</data></quarantine></set>'
```

An example of a successful response is shown below.

```
<response status="success"><result><iot-response>
  <version>2.0</version>
  <payload>
    <quarantine-add>
    </quarantine-add>
  </payload>
</iot-response>
</result></response>
```

List Quarantined Devices

Once you've added a device to the quarantine list on a firewall, you can use another XML API to access the updated list.

Make a request to get the list of quarantined devices from the firewall.

The following is an example of a curl command that requests a list of all the quarantined devices from a firewall.

```
curl -X POST 'https://<firewall>/api?key=<api
 key>&type=op&cmd=<request><device-quarantine-list><show></show></
device-quarantine-list></request>'
```

The following example shows the result format of the request.

```
<response status="success">
 <result>
  <entry name="12345abcde">
            <start>0</start>
            <hostid>12345abcde</hostid>
```



```
            <reason>Admin</reason>
            <source></source>
            <vsys_id>1</vsys_id>
            <serialno></serialno>
            <user></user>
            <timestamp>Tue Feb  4 15:48:32 2020</timestamp>
        </entry>
        <entry name="host3">
            <start>0</start>
            <hostid>host3</hostid>
            <reason>admin1</reason>
            <source>litest1</source>
            <vsys_id>1</vsys_id>
            <serialno>serial1</serialno>
            <user></user>
            <timestamp>Mon Apr 13 12:10:55 2020</timestamp>
        </entry>
        <total>2</total>
     </result>
</response>
```

Delete a Device From the Quarantine List

Management of the quarantine list on a firewall includes the ability to remove a device that is no longer compromised from the list.

Make an API request to delete a device from the quarantine list.

The following example shows a request to to delete a device**host3** from a firewall quarantine list.

```
curl -X POST 'https://<firewall>/api?key=<api
 key>&type=op&cmd=<request><device-quarantine-
list><delete><host>host3</host></delete></device-quarantine-list></
request>'
```

An example of a successful response is as follows.

```
<response status="success">
 <result>
     <status>success</status>
     <msg>Device is deleted from quarantine list</msg>
 </result>
</response>
```


Manage Certificates (API)

Using the XML API, you can automate the management workflow for certificates. You can programatically:

- Generate self-signed certificates
- Configure Certificate Authorities (CAs) to sign certificates
- Set certificates as Trusted Root CAs, Forward Trust Certificates, and Forward Untrust Certificates
- Renew and revoke certificates
- Bulk import and export certificates

For more information about the use of certificates on Palo Alto Networks Firewalls, see: Keys and Certificates.

STEP 1 | Send a request to generate a self-signed certificate.

With the XML API, you can generate certificates, flag the certificates as self-signed, and set cryptographic and certificate attributes in a single request.

The following example creates a certificate named SSCert with an IP address of 10.2.1.1 using RSA as the cryptographic algorithm. This certificate is set as a self-signed certificate using the element <ca> set to yes:

```
curl -X GET "<firewall>/api/?key

 <apikey>&type=op&cmd=<request><certificate><generate><algorithm><RSA><rsa-
nbits>512</rsa-nbits></RSA></algorithm><certificate-name>SSCert</
certificate-name><name>10.2.1.1</name><ca>yes</ca></generate></
certificate></request>"
```

STEP 2 | Send a request to set the certificate you created above as a trusted root certificate and a forward trust certificate.

The following requests use the configuration command and the xpath of the certificate you generated to set the certificate as a forward trust certificate and as a trusted root certificate.

```
curl -X GET "<firewall>/api/?
key=<apikey>&type=config&action=set&xpath=/config/shared/ssl-
decrypt&element=<trusted-root-CA><member>SSCert</member></trusted-
root-CA>"
```

```
curl -X GET "/api/?key=<apikey>&type=config&action=set&xpath=/
config/shared/ssl-decrypt&element=<forward-trust-
certificate><rsa>SSCert</rsa></forward-trust-certificate>"`
```


STEP 3 | Send a request to create a subordinate certificate using the self-signed certificate you generated.

The following request creates a subordinate of the SSCert that you can use to get more granular control in the chain of trust.

```
curl -X GET "<firewall>/api/?
key=<apikey>&type=op&cmd=<request><certificate><generate><algorithm><RSA><rsa
nbits>512</rsa-nbits></RSA></algorithm><certificate-
name>subordinate</certificate-name><name>subordinateip</
name><digest>sha256</digest><signed-by>SSCert</signed-by></
generate></certificate></request>"
```

STEP 4 | Send a request to export certificates locally so that you can install the certificates on your clients.

The following request downloads the self-signed certificate as SSCert.pem.

```
curl -o SSCert.pem "<firewall>/api/?
key=<apikey>&type=op&cmd=<download><certificate><certificate-
name>SSCert</certificate-name><format>pem</format></certificate></
download>"
```

STEP 5 | Import the certificates to other firewalls.

The following request uploads the SSCert certificate to a firewall.

```
curl -F "file=@<path of the file>" "<firewall>/api/?
key=<apikey>&type=import&category=certificate&certificate-
name=SSCert&format=pem"
```

Alternatively, to import both the certificate and private key to your firewalls at the same time, use the following command:

```
curl -F "file=@<path of the file>" "<firewall>/api?
key=<apikey>type=import&category=keypair&certificate-
name=SSCert.pem.txt&format=pem&passphrase=
        secretphrase
```

To import a certificate to a specific template and device on Panorama, use the following command:

```
curl -F "file=@<path of the file>" "<firewall>/api/?
key=<apikey>&type=import&category=certificate&certificate-
name=SSCert&format=pem&target-tpl=template&target-tpl-
vsys=vsys1"
```


STEP 6 | Renew and revoke certificates.

The following request revokes the subordinate certificates.

```
curl - X GET "<firewall>/api/?
key=<apikey>&type=op&cmd=<request><certificate><revoke><certificate-
name>subordinate</certificate-name></revoke></request></
certificate>"
```

The following request renews the self-signed root certificate that you generated.

```
curl - X GET "<firewall>/api/?
key=<apikey>&type=op&cmd=<request><certificate><renew><certificate-
name>SSCert</certificate-name><days-till-expiry>365</days-till-
expiry></renew></certificate></request>"
```

STEP 7 | Send a request to commit the changes.

```
curl - X GET "<firewall>/api/?type=commit&cmd=<commit></
commit>&key=<apikey>"
```


PAN-OS XML API Request Types

The following topics provide common request examples that you can use to better understand the PAN-OS XML API.

PAN-OS XML API Request Types and Actions

Use PAN-OS XML API to run various requests depending on the request type that you specify:

- Request Types
- Configuration Actions

Request Types

You can currently use the following request types:

Syntax	Description
type=keygen	Generate API keys for authentication.
type=config	Modify the configuration.
type=commit	Commit firewall configuration, including partial commits.
type=op	Perform operational mode commands, including checking system status and validating configurations.
type=report	Get reports, including predefined, dynamic, and custom reports.
type=log	Get logs, including traffic, threat, and event logs.
type=import	Import files including configurations and certificates.
type=export	Export files including packet captures, certificates, and keys.
type=user-id	Update User-ID mappings.
type=version	Show the PAN-OS version, serial number, and model number.

Configuration Actions

In addition to the request type that you specify, use available actions to modify or read configurations using**type=config**:

- Actions for Modifying a Configuration
- Actions for Reading a Configuration

Actions for Modifying a Configuration

Configuration Action Type	Syntax
Set candidate configuration	**action=set**
Edit candidate configuration	**action=edit**
Delete candidate object	**action=delete**
Rename a configuration object	**action=rename**
Clone a configuration object	**action=clone**
Move a configuration object	**action=move**
Override a template setting	**action=override**
Move multiple objects in a device group or virtual system	**action=multi-move**
Clone multiple objects in a device group or virtual system	**action=multi-clone**
Show available subnode values and XPaths for a given XPath.	**action=complete**

Set and edit actions differ in two important ways:

- Set actions add, update, or merge configuration nodes, while edit actions replace configuration nodes.
- Set actions are non-destructive and are only additive, while edit actions can be destructive.

Actions for Reading a Configuration

Configuration Action Type	Syntax
Get active configuration	**action=show**
Get candidate configuration	**action=get**

Show and get actions differ in three important ways:

- Show actions retrieve the active configuration, while get actions retrieve the candidate, uncommitted configuration.
- Show actions only work when the provided XPath specifies a single node. Get actions work with single and multiple nodes.

- Show actions can use relative XPath, while get actions require absolute XPath.

Asynchronous and Synchronous Requests to the PAN-OS XML API

Most PAN-OS XML API requests are synchronous, meaning the response immediately provides the requested data. For example, when you Make Your First API Call and request system information, the API response is immediate and contains information such as the IP address, hostname, and model of your firewall.

However, there are some Request Types that require more time to process and are asynchronous, meaning they require more than one request to get final results. These API requests include the following:

- Get Reports (API)
- Retrieve Logs (API)
- Export Technical Support Data
- Some requests to Run Operational Mode Commands (API), including download, upgrade, and installation requests

With asynchronous requests, you first initiate a request. The API responds with a job ID while it processes your request. In your subsequent requests, you use this job ID to check on the results of your original request.

Examples (replace `jobid` with the actual job ID):

- Get reports:

```
https://<firewall>/api/?key=
      apikey&type=report&action=get&job-id=jobid
```

- Retrieve logs:

```
https://<firewall>/api/?key=
      apikey&type=log&action=get&job-id=jobid
```

- Export technical support data:

```
https://<firewall>/api/?key=
      apikey&type=export&category=tech-support&action=get&job-
id=jobid
```

- Commit:

```
https://<firewall>/api/?key=
      apikey&type=commit&cmd=<commit></commit>
```

- Operational commands:

```
https://<firewall>/api/?key=
```



```
apikey&type=op&cmd=<show><jobs><id>jobid</id></jobs></show>
```


Configuration (API)

The requests examples in these topics illustrate how you can use the PAN-OS XML API to configure your firewall.

- Get Active Configuration
- Get Candidate Configuration
- Set Configuration
- Edit Configuration
- Delete Configuration
- Rename Configuration
- Clone Configuration
- Move Configuration
- Override Configuration
- Multi-Move or Multi-Clone Configuration
- Multi-config Request (API)
- View Configuration Node Values for XPath

Get Active Configuration

- Use XPath to Get Active Configuration
- Use XPath to Get ARP Information

Use XPath to Get Active Configuration

Use **action=show** with no additional parameters to retrieve the entire active configuration.

STEP 1 | Use the **xpath** parameter to target a specific portion of the configuration. For example, to retrieve just the security rulebase: **xpath=/config/devices/entry/vsys/entry/rulebase/security**:

```
curl -X POST 'https://firewall/api?type=config&action=show&xpath=/
config/devices/entry/vsys/entry/rulebase/security"
```

There is no trailing backslash character at the end of the XPath.

STEP 2 | Confirm that the XML response for the query looks similar to the following (truncated):

```
     <response status="success">
  <result>
    <security>
      <rules><entry name="IT DNS Services"><profile-
setting><group><member>best-practice</member></
group></profile-setting><to><member>untrust</member></
to><from><member>trust</member></from><source><member>any</
```



```
member></source><destination><member>Data Center</
member></destination><source-user><member>any</
member></source-user><category><member>any</
member></category><application><member>dns</member></
application><service><member>application-default</
member></service><hip-profiles><member>any</member></hip-
profiles><action>allow</action><tag><member>Best Practice</
member></tag><log-start>no</log-start><log-setting>default</log-
setting></entry>
       ...
            </rules>
    </security>
  </result>
</response>
```

Use XPath to Get ARP Information

Follow this procedure to use XPath to Get ARP Information.

STEP 1 | Use the following request to retrieve ARP information:

```
https://<firewall>//api/?type=op&command=<show><arp><entry
 name='all'/></arp></show>
```

STEP 2 | Confirm that the XML response for the query looks like the following (truncated):

```
<response status="success">
 <result>
 <max>3000</max>
 <total>16</total>
 <timeout>1800</timeout>
 <dp>dp0</dp>
 <entries>
 <entry>
 <status>c</status>
 <ip>10.47.0.1</ip>
 <mac>00:1b:17:00:2f:13</mac>
 <ttl>1743</ttl>
 <interface>ethernet1/1</interface>
 <port>ethernet1/1</port>
 </entry>
 <entry>
 <status>c</status>
 <ip>10.47.0.10</ip>
 <mac>00:50:56:93:68:6f</mac>
 <ttl>386</ttl>
 <interface>ethernet1/1</interface>
 <port>ethernet1/1</port>
 </entry>
 <!-- truncated -->
 </result>
```



```
        </response>
```

Get Candidate Configuration

Get the candidate configuration from a firewall by specifying the portion of the configuration to get. Use the following request, including the**xpath** parameter to specify the portion of the configuration to get.

```
curl -X POST 'https://firewall/api?
type=config&action=get&xpath=<path-to-config-node>"
```

Configuration Node	API Request
Firewall candidate configuration	`curl -X POST 'https://firewall/ api?type=config&action=get&xpath=/ config/devices/entry/vsys/ entry[@name='vsys1']&key=<api_key>"`
Firewall candidate configuration through Panorama	`curl -X POST 'https://panorama/ api?type=config&action=get&xpath=/ config/devices/entry/vsys/ entry[@name='vsys1']&target=<serial>&key=<panorama_api_k`
Firewall candidate configuration through Panorama without specifying a firewall	`curl -X POST 'https://panorama/ api?type=config&action=get&xpath=/ config/devices/entry/*[name()! ='vsys']|/config/devices/entry/vsys/ entry[@name='vsys1']&key=<panorama_api_key>"`

Address objects in a virtual system (vsys).	`curl -X GET "https://<firewall>//api/? type=config&action=get&xpath=/config/devices/ entry/vsys/entry[@name='vsys1']/address"`
	The response looks similar to the following:
	`<response status="success" code="19"> <result total-count="1" count="1"> <address admin="name" dirtyId="8" time="2015/10/20 15:32:36"><entry name="testobject"><ip-netmask>192.0.2.2</ ip-netmask></entry><entry name="test1"><ip- netmask>192.0.2.12</ip-netmask></entry>`

Configuration Node	API Request
	...</address> </result> </response>
Pre-rules pushed from Panorama.	**curl -X GET "https://<firewall>//api/?type=config&action=get&xpath=/config/panorama/vsys/entry[@name='vsys']/pre-rulebase/security"**
Full list of all applications.	**curl -X POST 'https://firewall/api?type=config&action=get&xpath=/config/predefined/application"**
Details on the specific application.	**curl -X POST 'https://firewall/api?type=config&action=get&xpath=/config/predefined/application/entry[@name='hotmail']"**

Set Configuration

Use **action=set** to add or create a new object at a specified location in the PAN-OS configuration. Use the **xpath** parameter to specify the location of the object in the configuration. For example, if you are adding a new rule to the security rulebase, the xpath-value would be:

```
/config/devices/entry[@name='localhost.localdomain']/vsys/entry[@name='vsys1']/rulebase/security
```

Use the **element** parameter to specify a value for the object you are adding or creating using XML.

Configuration Node	API Request
Create a new rule called rule1 in security policy	**curl -X POST 'https://firewall/api?type=config&action=set&key=keyvalue&xpath=xpath-value&element=element-value"**

Configuration Node	API Request

	where the xpath-value is:
	`/config/devices/entry/vsys/entry/rulebase/security/rules/entry[@name='rule1']`
	and the element-value is:
	`<source><member>src</member></source><destination><member>dst</member></destination><service><member>service</member></service><application><member>application</member></application><action>action</action><source-user><member>src-user</member></source-user><option><disable-server-response-inspection>yes-or-no</disable-server-response-inspection></option><negate-source>yes-or-no</negate-source><negate-destination>yes-or-no</negate-destination><disabled>yes-or-no</disabled><log-start>yes-or-no</log-start><log-end>yes-or-no</log-end><description>description</description><from><member>src-zone</member></from><to><member>dst-zone</member></to>`

Add an additional member to an address group or list	Include the 'list' node in the xpath using the`member[text()='name']` syntax and include the members in the element parameter. For example, to add an additional static address object named`abc` to an address group named`test`, use: `curl -X POST 'https://firewall/api?type=config&action=set&xpath=/config/devices/entry/vsys/entry[@name='vsys1']/address-group/entry[@name='test']&element=<static><member>abc</member></static>"`
Create a new IP address on a specific interface	Specify the interface and IP address in the request: `curl -X GET "https://<firewall>/api?type=config&action=set&xpath=/config/devices/entry[@name='localhost.localdomain']/network/interface/ethernet/entry[@name='ethernet1/1']/layer3/ip&element=<entry name='5.5.5.5/24'/>"`

Configuration Node	API Request
Enable or disable a security rule	`curl -X POST 'https://firewall/api?type=config&action=set&xpath=/config/devices/entry[@name='localhost.localdomain']/vsys/entry[@name='<vsys1>']/rulebase/security/rules/entry[@name='<rule-name>']&element=<disabled>yes</disabled>"` Altenatively, use **<disabled>no</disabled>** to enable a rule.

Edit Configuration

Use **action=edit** to replace an existing object hierarchy at a specified location in the configuration with a new value. Use the xpath parameter to specify the location of the object, including the node to be replaced. Use the element parameter to specify a new value for the object using its XML object hierarchy (as seen in the output of **action=show**).

STEP 1 | Replace the application(s) currently used in a rule rule1 with a new application:

```
curl -X POST 'https://firewall/api?type=config&action=edit&xpath=xpath-value&element=element-value"
```

where

```
xpath=/config/devices/entry/vsys/entry/rulebase/security/rules/entry[@name='rule1']/application&element=<application><member>app-name</member></application>
```

STEP 2 | Use the response from the config show API request to create the XML body for the element.

```
curl -X POST 'https://firewall/api?type=config&action=show"
```

STEP 3 | Optionally replace all members in a node with a new set of members using the entry tag in both the xpath and element parameters. For example, to replace all the address objects in the address group named test with two new static members named **abc** and **xyz**, use:

```
curl -X POST 'https://firewall/api?type=config&action=edit&xpath=/config/devices/entry/vsys/entry[@name='vsys1']/address-group/entry[@name='test']&element=<entry
 name='test'><static><member>abc</member><member>xyz</member></static></entry>"
```


Delete Configuration

Use **action=delete** to delete an object at a specified location in the configuration. Use the **xpath** parameter to specify the location of the object to be deleted.

To delete more than one object using a config type xml command, create individual requests and repeat the process for each object.

- Delete a rule named rule1 in the security policy:

```
curl -X POST 'https://firewall/api?
type=config&action=delete&xpath=/config/devices/entry/vsys/entry/
rulebase/security/rules/entry[@name='rule1']"
```

- Delete a single member object in a group, use the object name in the xpath as **member[text()='name']**. For example, to delete a static address object named abc in an address group named test, use the following xpath:

```
curl -X POST 'https://firewall/api?
type=config&action=delete&xpath=/config/devices/entry/vsys/
entry[@name='vsys1']/address-group/entry[@name='test']/static/
member[text()='abc']"
```

Rename Configuration

Use **action=rename** to rename an object at a specified location in the configuration. Use the xpath parameter to specify the location of the object to be renamed. Use the newname parameter to provide a new name for the object.

STEP 1 | Use the following API query to rename an address object called **old_address** to **new_address**:

```
curl -X POST 'https://firewall/api?
type=config&action=rename&xpath=/config/
devices/entry/vsys/entry[@name='vsys1']/address/
entry[@name='old_address']&newname=new_address"
```

STEP 2 | Confirm that the XML response for the request looks like the following:

```
     <response status="success" code="20">
  <msg>command succeeded</msg>
</response>
```


Clone Configuration

Use **action=clone** to clone an existing configuration object. Use the **xpath** parameter to specify the location of the object to be cloned. Use the **from** parameter to specify the source object, and the **newname** parameter to provide a name for the cloned object.

STEP 1 | Use the following API query to clone a security policy called rule1 to rule2:

```
curl -X POST 'https://firewall/api?type=config&action=clone&xpath=/
config/devices/entry/vsys/entry[@name='vsys1']/rulebase/security/
rules&from=/config/devices/entry/vsys/entry[@name='vsys1']/
rulebase/security/rules/entry[@name='rule1']&newname=rule2"
```

STEP 2 | Confirm that the XML response for the request looks like the following:

```
     <response status="success" name="rule2"/>
```

A corresponding success log is recorded in the Configuration log:

```
     1,2014/03/19 19:07:45,0009C100708,CONFIG,0,0,2014/03/19
      19:07:45,10.66.18.1,,clone,admin,Web,Succeeded, config
devices entry vsys
      vsys1 rulebase security rules,384,0x8000000000000000
```

Move Configuration

Use **action=move** to move the location of an existing configuration object. Use the **xpath** parameter to specify the location of the object to be moved, the **where** parameter to specify type of move, and **dst** parameter to specify the destination path.

- **where=after&dst=xpath**
- **where=before&dst=xpath**
- **where=top**
- **where=bottom**

STEP 1 | Use the following API query to move a security policy called **rule1** to come after **rule2**:

```
curl -X POST 'https://firewall/api?type=config&action=move&xpath=/
config/devices/entry/vsys/entry[@name='vsys1']/rulebase/security/
rules/entry[@name='rule1']&where=after&dst=rule2"
```

STEP 2 | Confirm that the XML response for the request looks like the following:


```
      <response status="success" code="20">
  <msg>command succeeded</msg>
</response>
```

Override Configuration

Use**action=override** to override a setting that was pushed to a firewall from a template. Use the**xpath** parameter to specify the location of the object to override.

STEP 1 | Override the SNMP Trap profile configuration settings that were pushed to the firewall using a template:

```
curl -X POST 'https://firewall/api?
type=config&action=override&xpath=/config/shared/log-settings/
snmptrap&element=<entry name="snmp" src="tpl"><version
 src="tpl"><v2c src="tpl"><server src="tpl"><entry name="test"
 src="tpl"><manager src="tpl">2.2.2.2</manager><community
 src="tpl">test</community></entry></server></v2c></version></
entry>"
```

STEP 2 | Confirm that the XML response for the request looks like the following:

```
      <response status="success" code="20">
  <msg>command succeeded</msg>
</response>
```

Multi-Move or Multi-Clone Configuration

Use the**action=multi-move** and**action=multi-clone** actions to move and clone addresses, address groups, services, and more across device groups and virtual systems. Templates do not support the multi-move and multi-clone capability.

The syntax for multi-move and multi-clone specifies the xpath for the destination where the addresses will be moved to, the xpath for the source and the list of objects within the specified source. It also includes a flag for displaying the errors when the firewall performs a referential integrity check on the multi-move or multi-clone action.

- Move addresses**addr1**,**addr2**, to device group**norcal** from device group**socal**:

```
curl -X POST 'https://firewall/api?type=config&action=multi-
move&xpath=/config/devices/entry[@name='localhost.localdomain']/
devicegroup/entry[@name='norcal']/address&element=<selected-
list><source xpath="/config/devices/
entry[@name='localhost.localdomain']/devicegroup/
entry[@name='socal']/address"><member>addr1</member><member>addr2</
member></source></selected-list><all-errors>no</all-errors>"
```


- Clone addresses**addr1**,**addr2**, to device group**norcal** from device group**socal**:

```
curl -X POST 'https://firewall/api?type=config&action=multi-
clone&xpath=/config/devices/entry[@name='localhost.localdomain']/
devicegroup/entry[@name='norcal']/address&element=<selected-
list><source xpath="/config/devices/
entry[@name='localhost.localdomain']/devicegroup/
entry[@name='socal']/address"><member>addr1</member><member>addr2</
member></source></selected-list><all-errors>no</all-errors>"
```

Multi-config Request (API)

The PAN-OS XML API provides a mechanism to perform multiple configuration API requests within a single transaction. This can be useful to simplify multi-step API configurations.

Below is an example of how a typical multi-config request is structured:

```
<multi-config>
  <action id="action-id" xpath="xpath">
      <element-xml>
      </element-xml>
  </action>
  <action2 id="action2-id" xpath="xpath">
      <element-xml>
      </element-xml>
  </action2>
  <actionN id="actionN-id" xpath="xpath">
      <element-xml>
      </element-xml>
  </actionN>
</multi-config>
```

The root element, which is denoted above by the multi-config element above, can be any name. The actions can be any of the listed Actions for Modifying a Configuration except complete. The ID attribute is optional and you can use the ID to identify the specific responses to a single node in the request.

This response corresponds with the example above: `<response status="success" code="20"><response status="success" code="20" id="action-id"><msg>command succeeded</msg></response><response status="success" code="20" id="action2-id"><msg>command succeeded</msg></response></response>`

Besides xpath, other attributes are acceptable, for example: newname for the rename action. The request accepts the element-xml document if you can use an element argument for that particular request.

Consider the following when using the multi-config request type:

- *When a request in the multi-config operation fails, no configuration changes are performed.*
- *You can add additional measures by adding the parameter* `strict-transactional=yes`. *When set:*
  - *When a commit operation is active or a commit is pending, the operation will fail.*
  - *When there are uncommitted changes for the user performing the operation, they will be rolled back before performing the multi-config operation.*

Example

To use the multi-config XML-API, you can URL encode an XML document and send it to the firewall. For example this document is saved as multi-config.xml:

```
<multi-config>
  <set id="101" xpath="/config/devices/
entry[@name='localhost.localdomain']/vsys/entry[@name='vsys1']/
address/entry[@name='addr10']">
    <ip-netmask>10.0.0.10</ip-netmask>
  </set>
  <set id="102" xpath="/config/devices/
entry[@name='localhost.localdomain']/vsys/entry[@name='vsys1']/
address-group/entry[@name='group1']/static">
    <member>addr10</member>
  </set>
</multi-config>
```

This file, can be sent using the following cURL request:

```
$ curl -ku user:pass https://firewall/api -d type=config -d
 action=multi-config \
> --data-urlencode element@multi-config.xml
```

A successful response returns:

```
<response status="success" code="20"><response status="success"
 code="20" id="101"><msg>command succeeded</msg></response><response
 status="success" code="20" id="102"><msg>command succeeded</msg></
response></response>
```

If you set the `strict-transactional=yes` parameter you may get a different response if there is a commit in progress.

```
$ curl -ku user:pass https://firewall/api -d type=config
 -d action=multi-config -d action=multi-config -d strict-
transactional=yes\
> --data-urlencode element@multi-config.xml
```


If you send the above command while another commit is in progress you may receive the following response:

```
<response status="error" code="15"><msg><line>Commit in Progress or
 Pending</line></msg></response>
```

View Configuration Node Values for XPath

Use**action=complete** action along with an XPath to see possible values that are available with the XPath node.

STEP 1 | View the possible values, such as network interfaces, for multi-vsys firewalls, use the following command:

```
curl -X POST 'https://firewall/api?
type=config&action=complete&xpath=/config/devices/
entry[@name='localhost.localdomain']/vsys&key=apikey"
```

STEP 2 | Confirm that the XML response for the request looks like the following:

```
     <response status="success" code="19">
  <completions>
    <completion value="vsys1" vxpath="/config/devices/
entry[@name='localhost.localdomain']/vsys/entry[@name='vsys1']"
 current="yes" help-string="vsys1"/>
  </completions>
</response>
```


Commit Configuration (API)

You can use the commit API request to commit a candidate configuration to a firewall.

You can validate or revert a candidate configuration before committing it using Run Operational Mode Commands (API).

- Commit
- Commit-All

Commit

Replace the**body** element in the**cmd** parameter with the XML element for the corresponding commit operation.

Use the API Browser *to find different options available for use with force and partial commits.*

STEP 1 | Use one of the following requests to commit a configuration:

- **Commit**— Commit candidate changes to the firewall.

```
curl -X POST 'https://firewall/api?type=commit&cmd=<commit></
commit>"
```

- **Force Commit**—

```
curl -X POST 'https://firewall/api?
type=commit&cmd=<commit><force></force></commit>"
```

- **Partial commit while excluding shared objects and device and network configuration**—

```
curl -X POST 'https://firewall/api?
type=commit&action=partial&cmd=<commit><partial><device-and-
network>excluded</device-and-network><shared-object>excluded</
shared-object></partial></commit>"
```

- **Partial commit admin-level changes**— To commit admin-level changes on a firewall, include the administrator name in the request.

```
curl -X POST 'https://firewall/api?
&type=commit&action=partial&cmd=<commit><partial><admin><member>
$admin-name</member></admin></partial></commit>"
```


- **Partial commit admin-level changes on a firewall or Panorama while excluding shared objects**—Include the administrator name in the request. Replace the $admin-name Replace

```
curl -X POST 'https://firewall/api?
&type=commit&action=partial&cmd=<commit><partial><device-and-
network>excluded</device-and-network><shared-object>excluded</
shared-object><admin><member>socadmin</member></admin></partial></
commit>"
```

STEP 2 | Confirm that the XML response indicates that there were no changes to commit or that the changes are queued for commit:

- No pending changes to commit:

```
        <response status="success" code="19">

           <msg>There are no changes to commit.</msg></response>
```

- Pending changes:

```
<response status="success" code="19">
    <result>
        <msg>
            <line>Commit job enqueued with jobid 4</line>
        </msg>
        <job>4</job>
    </result>
</response>
```

STEP 3 | Query the status of the job using the job ID:

```
curl -X POST 'https://firewall/api?type=op&cmd=<show><jobs><id>4</
id></jobs></show>"
```

STEP 4 | Confirm that the XML response details state the Configuration was committed successfully:

```
<response status="success">
    <result>
        <job>
            <tenq>2021/07/21 14:33:55</tenq>
            <tdeq>14:33:55</tdeq>
            <id>4</id>
            <user>admin</user>
            <type>Commit</type>
```



```
            <status>ACT</status>
            <queued>NO</queued>
            <stoppable>yes</stoppable>
            <result>PEND</result>
            <tfin></tfin>
            <description></description>
            <positionInQ>0</positionInQ>
            <progress>55</progress>
            <warnings>
                <line>Aggregate-ethernet interface ae1 has no
 member interfaces.</line>
            </warnings>
            <details></details>
        </job>
    </result>
</response>
```

Commit-All

To centrally manage firewalls from Panorama, use the commit-all API request type to push and validate shared policy to the firewalls using device groups and configuration to Log Collectors and firewalls using templates or template stacks.

Commit Type	API Request
Pre-commit policy validation.	`curl -X POST 'https://panorama/api?type=commit&action=all&cmd=<commit-all><shared-policy><validate-only></validate-only></shared-policy></commit-all>"`
Specific device group commit.	`curl -X POST 'https://panorama/api?type=commit&action=all&cmd=<commit-all><shared-policy><device-group><entry name="<device-group-name>"/></device-group></shared-policy></commit-all>"`
Specific device group commit without including default device/network template changes.	`curl -X POST 'https://panorama/api?type=commit&action=all&cmd=<commit-all><shared-policy><include-template>no</include-template><device-group><entry name="<device-group-name>"/></device-group></shared-policy></commit-all>"`
Virtual system (vsys) commit.	`curl -X POST 'https://panorama/api?type=commit&action=all&cmd=<commit-all><shared-policy><device-`

Commit Type	API Request
	`group><entry name="<device-group-name>"/ ><devices><entry name="<serial_number>"><vsys><member>vs name</member></vsys></entry></devices></ device-group></shared-policy></commit-all>"`
Specific firewall commit.	`curl -X POST 'https://panorama/api? type=commit&action=all&cmd=<commit- all><shared-policy><device- group><entry name="<device-group- name>"><devices><entry name="<serial_number>"></ devices></entry></device-group></shared- policy></commit-all>"`

Use the API Browser to find other options available for granular commit operations on Panorama. In the **cmd** parameter, you must replace the XML element for the corresponding **commit-all** operation.

Run Operational Mode Commands (API)

Use any of the operational mode commands available on the command line interface with the following API request:

```
curl -X POST 'https://firewall/api?type=op&cmd=<xml-body>"
```

Use the API Browser to explore operational mode commands and a complete listing of all the options available for the**xml-body** and their corresponding operation.

Some requests operational mode commands, including download, upgrade, and installation requests, are asynchronous, meaning they require more than one request to get final results. Learn more about Asynchronous and Synchronous Requests to the PAN-OS XML API.

Operational Command	API Request
System restart.	`curl -X POST 'https://firewall/api?type=op&cmd=<request><restart><system></system></restart></request>"`
System software version installation.	`curl -X POST 'https://firewall/api?type=op&cmd=<request><system><software><install><version version></install></software></system></request>"`
Multi-vsys mode.	`curl -X POST 'https://firewall/api?type=op&cmd=<set><system><setting><multi-vsys></multi-vsys></setting></system></set>"`
User Activity Report scheduling.	`curl -X POST 'https://firewall/api?type=op&cmd=<schedule><uar-report><user>username</user><title>titlename</title></uar-report></schedule>"`
Detailed information on applications and threats from the firewall.	`curl -X POST 'https://firewall/api?type=op&cmd=<show><predefined><xpath>/predefined/threats/vulnerability/`

Operational Command	API Request
	`entry[@name='30003']</xpath></predefined></show>"`
Full configuration validation.	`curl -X POST 'https://firewall/api?type=op&cmd=<validate><full></full></validate>"`
Partial configuration validation.	`curl -X POST 'https://firewall/api?type=op&cmd=<validate><partial><device-and-network>excluded</device-and-network></partial></validate>"`
Configuration saving.	`curl -X POST 'https://firewall/api?type=op&cmd=<save><config><to>filename</to></config></save>"`
Configuration loading.	`curl -X POST 'https://firewall/api?type=op&cmd=<load><config><from>filename</from></config></load>"`
Partial revert of admin-level changes for a candidate configuration on a firewall.	`curl -X POST 'https://firewall/api?type=op&cmd=<revert><config><partial><admin><member>adminame</member></admin></partial></config></revert>"`
Partial revert of admin-level changes to Panorama by a specific administrator within a specific device group	`curl -X POST 'https://panorama/api?type=op&cmd=<revert><config><partial><admin><member><admname></member></admin><device-group><member><device-group-name></member></device-group><no-template/><no-template-stack/><no-log-collector-group/><no-log-collector/><device-and-network>excluded</device-and-network></partial></config></revert>"`
Base64-encoded metadata of a SAML authentication profile.	`curl -X POST 'https://firewall/api?type=op&cmd=<show><sp-metadata><management><authprofile><SAML-auth-`

Operational Command	API Request
	profile-name></authprofile></management></sp-metadata></show>"
Summary of changes between the active and candidate configuration.	**curl -X POST 'https://firewall/api?type=op&cmd=<show><config><list><change-summary/></list></config></show>"**
Commit locks	**curl -X POST 'https://firewall/api?type=op&cmd=<show><commit-locks/></show>"**
Remove configuration lock	**curl -X POST 'https://firewall/api?type=op&cmd=<request><config-lock><remove></remove><config-lock><remove></config-lock/></request>"** To remove the configuration lock for a particular Device Group, append &vsys=TargetDG to the end of request where TargetDG is the name of the Device Group.

Remove configuration locks for a particular template	1. Change the target template using the following command:
	`curl -X POST 'https://firewall/api?type=op&cmd=<set><system><setting><target><template><nTemplate</name></template></target></setting></system></set>`
	2. Issue the remove configuration lock command:
	`curl -X POST 'https://firewall/api?type=op&cmd=<request><config-lock><remove></remove></config-lock></request>`

Show WildFire appliances connected to Panorama.	**curl -X POST 'https://panorama/api?&type=op&cmd=<show><wildfire-appliance><connected></connected></wildfire-appliance></show>"**
System summary about WildFire appliances or WildFire clusters.	• **WildFire Appliance**: **curl -X POST 'https://panorama/api?&type=op&cmd=<show><wildfire-appliance><all></all></wildfire-appliance></show>"**

Operational Command	API Request
	• **WildFire Cluster:** `curl -X POST 'https://panorama/api?&type=op&cmd=<show><wildfire-appliance-cluster><all></all></wildfire-appliance-cluster></show>"`

Generate a list of Firewalls connected and sending data to a WildFire appliance or WildFire cluster.	• **WildFire Appliance:**
	`curl -X POST 'https://panorama/api?&type=op&cmd=<show><wildfire-appliance><devices-reporting-data><serial-number><serial_number></serial-number></devices-reporting-data></wildfire-appliance></show>"`
	• **WildFire Cluster:**
	`curl -X POST 'https://panorama/api?&type=op&cmd=<show><wildfire-appliance-cluster><devices-reporting-data><name><cluster_name></name></devices-reporting-data></wildfire-appliance-cluster></show>"`

Display configuration details about a specified WildFire appliance or WildFire cluster.	• **WildFire Appliance:**
	`curl -X POST 'https://panorama/api?&type=op&cmd=<show><wildfire-appliance><info><serial-number><serial_number></serial-number></info></wildfire-appliance></show>"`
	• **WildFire Cluster:**
	`curl -X POST 'https://panorama/api?&type=op&cmd=<show><wildfire-appliance-cluster><info><name><cluster_name></name></info></wildfire-appliance-cluster></show>"`

Display registration activity for a specified WildFire appliance or WildFire cluster.	• **WildFire Appliance:** `curl -X POST 'https://panorama/api?&type=op&cmd=<show><wildfire-appliance><last-device-registration><all><serial-number><serial_number></serial-number></all></last-device-registration></wildfire-appliance></show>"`

Operational Command	API Request
	• **WildFire Cluster:** `curl -X POST 'https://panorama/api?&type=op&cmd=<show><wildfire-appliance-cluster><last-device-registration><all><name><cluster_name></name></all></last-device-registration></wildfire-appliance-cluster></show>"`

Display statistics for a specified WildFire appliance or WildFire cluster.	• **WildFire Appliance:**
	`curl -X POST 'https://panorama/api?&type=op&cmd=<show><wildfire-appliance><statistics><days><days_up_to_31></days><type><all_or_file_or_general></type><serial-number><serial_number></name></statistics></wildfire-appliance></show>"`
	• **WildFire Cluster:**
	`curl -X POST 'https://panorama/api?&type=op&cmd=<show><wildfire-appliance-cluster><statistics><hours><hours_up_to_24></minutes><type><all_or_file_or_general></type><name><cluster_name></name></statistics></wildfire-appliance-cluster></show>"`

Display a list of supported VM images on the specified WildFire appliance.	`curl -X POST 'https://panorama/api?&type=op&cmd=<show><wildfire-appliance><vm-images><serial-number><serial_number></serial-number></vm-images></wildfire-appliance></show>"`

Get Reports (API)

The XML API provides a way to quickly pull the results of any report defined in the system using the**type=report** parameter.

You can access three kinds of reports:

- Dynamic Reports (ACC reports)—**reporttype=dynamic**
- Predefined Reports—**reporttype=predefined**
- Custom Reports—**reporttype=custom**

To retrieve a specific report by name, use the**reportname** parameter:

```
curl -X POST 'https://firewall/api?type=report&reporttype=dynamic|
predefined|custom&reportname=<name>"
```

When you request a report, the API responds asynchronously with a job ID, which you can use to retrieve the reports. Learn more about Asynchronous and Synchronous Requests to the PAN-OS XML API.

- Dynamic Reports
- Predefined Reports
- Custom Reports

Dynamic Reports

You can use the API to view a number of dynamic reports, such as**top-applications-summary**,**top-blocked-url-summary**, and**top-spyware-threats-summary**. For dynamic reports, provide either a specific period using the**period** or a time frame using**starttime** and**endtime** options (use a + instead of a space between the date and timestamp). Use**topn** to determine the number of rows.

Dynamic Report Type	API Request
Full dynamic report list.	`curl -X POST 'https://firewall/api?type=report&reporttype=dynamic"`
Last 60 seconds.	`curl -X POST 'https://firewall/api?type=report&reporttype=dynamic&reportname=top-app-summary&period=last-60-seconds&topn=5"`

Dynamic Report Type	API Request
Last 15 minutes.	`curl -X POST 'https://firewall/api?type=report&reporttype=dynamic&reportname=top-app-summary&period=last-15-minutes&topn=5"`
Last hour.	`curl -X POST 'https://firewall/api?type=report&reporttype=dynamic&reportname=top-app-summary&period=last-hour&topn=5"`
Last 12 hours.	`curl -X POST 'https://firewall/api?type=report&reporttype=dynamic&reportname=top-app-summary&period=last-12-hrs&topn=5"`
Last calendar day.	`curl -X POST 'https://firewall/api?type=report&reporttype=dynamic&reportname=top-app-summary&period=last-calendar-day&topn=5"`
Last 7 days	`curl -X POST 'https://firewall/api?type=report&reporttype=dynamic&reportname=top-app-summary&period=last-7-days&topn=5"`
Last 7 calendar days	`curl -X POST 'https://firewall/api?type=report&reporttype=dynamic&reportname=top-app-summary&period=last-hour&topn=5"`
Last calendar week.	`curl -X POST 'https://firewall/api?type=report&reporttype=dynamic&reportname=top-app-summary&period=last-calendar-week&topn=5"`
Last 30 days	`curl -X POST 'https://firewall/api?type=report&reporttype=dynamic&reportname=top-app-summary&period=last-30-days&topn=5"`

Predefined Reports

Predefined reports always return data for the last 24-hour period. You can also get this list by following the link for predefined reports, such as**top-applications**,**top-attackers**, and**bandwidth-trend** on the API browser.

Dynamic Report Type	API Request
Full predefined report list.	`curl -X POST 'https://firewall/api?type=report&reporttype=predefined"`
Top applications.	`curl -X POST 'https://firewall/api?type=report&async=yes&reporttype=predefined&reportname=t application-categories"`
Top attackers.	`curl -X POST 'https://firewall/api?type=report&async=yes&reporttype=predefined&reportname=t attackers"`
Top victims.	`curl -X POST 'https://firewall/api?type=report&async=yes&reporttype=predefined&reportname=t victims"`

Custom Reports

For custom reports, the selection criteria, such as time frame, group-by, and sort-by are part of the report definition. The API returns any shared custom reports. Note that quotes are not required around the report name and any spaces in the report name must be URL encoded to**%20**.

For custom reports created in a specific VSYS, you can retrieve them directly by specifying the**vsys** parameters.

STEP 1 | Retrieve the report definition from the configuration:

```
curl -X POST 'https://firewall/api?type=config&action=get&xpath=/
config/devices/entry/vsys/entry[@name='vsys1']/reports/
entry[@name='report-abc']"
```

STEP 2 | Create a job to retrieve a dynamic report using**reporttype=dynamic**,**reportname=custom-dynamic-report**,

and **cmd=report-definition** where **report-definition** is the XML definition retrieved in the previous query:

```
curl -X POST 'https://firewall/api?
type=report&reporttype=dynamic&reportname=custom-dynamic-
report&cmd=<type><appstat><aggregate-by><member>category-of-
name</member><member>technology-of-name</member></aggregate-
by></appstat></type><period>last-24-hrs</period><topn>10</
topn><topm>10</topm><query>(name+neq+'')AND(vsys+eq+'vsys1')</
query>"
```

The response includes the job ID you can use to view the results:

```
<response status="success">
  <result>
    <msg>
      <line>Report job enqueued with jobid 6</line>
    </msg>
    <job>6</job>
  </result>
</response>
```

STEP 3 | View the dynamic report:

```
curl -X POST 'https://firewall/api?type=report&action=get&job-
id=jobid"
```


Export Files (API)

You can export certain types of files from the firewall using the**type=export** parameter in the API request.

Use the category parameter to specify the type of file that you want to export.

- Configuration—**category=configuration**
- Certificates/Keys—**category=certificate**
- Response pages—**category= application-block-page | captive-portal-text | file-block-continue-page | file-block-page | global-protect-portal-custom-help-page | global-protect-portal-custom-login-page | global-protect-portal-custom-welcome-page | ssl-cert-status-page | ssl-optout-text | url-block-page | url-coach-text | virus-block-page>**
- Technical support data—**category=tech-support**
- Device State—**category=device-state**

Use cURL tools to export the file from the firewall and save locally with a local file name:

```
curl -o
    <filename> "https://<firewall>/api/??key=apikey&<query-
parameters>"
```

When using the API query from a web browser, you can specify**to=filename** as an optional parameter if you would like to provide a different name when saving the file locally.

- Export Packet Captures
- Export Certificates and Keys
- Export Technical Support Data

Export Packet Captures

You can export packet captures from the firewall by specifying the PCAP type using the**category** parameter:

- Export Application PCAPS
- Export Threat, Filter, and Data Filtering PCAPs

Export Application PCAPS

Application PCAPs are organized by a directory/filename structure where the directory is a date in**yyyymmdd** format. Filenames for application pcaps use a**SourceIP-SourcePort-DestinationIP-DestinationPort-SessionID.pcap** format.

Application PCAP Type	API Request
Application PCAP directory list.	`curl -X POST 'https://firewall/api?type=export&category=application-pcap"`
List of files under a directory using the**from** parameter to indicate date.	`curl -X POST 'https://firewall/api?type=export&category=application-pcap&from=<yyyymmdd>"`
Application PCAP file by name using the**from** parameter.	`curl -X POST 'https://firewall/api?type=export&category=application-pcap&from=<yyyymmdd>/<filename>"` The file will be retrieved and saved locally using the name yyyymmdd-filename.
Application PCAP file saved locally with a custom name using the**to** parameter.	`curl -X POST 'https://firewall/api?type=export&category=application-pcap&from=<yyyymmdd>/<filename>&to=<localfile>"`

Export Threat, Filter, and Data Filtering PCAPs

To export threat PCAPs, you need to provide the PCAP ID from the threat log and the search time, which is the time that the PCAP was received on the firewall. Threat PCAP filenames use a**pcapID.pcap** format.

PCAP Type	API Request
Threat PCAP using PCAP ID, device name, session ID, and search	`curl -X POST 'https://firewall/api?type=export&category=threat-pcap&pcap-id=<id>&device_name=<device name>&sessionid=<session id>&search-time=<yyyy/mm/dd+hr:min:sec>"`

PCAP Type	API Request
List of filtered PCAPs	`curl -X POST 'https://firewall/api?type=export&category=filters-pcap"`
Specific filtered PCAP file	`curl -X POST 'https://firewall/api?type=export&category=filters-pcap&from=<filename>"`
List of data filtering PCAP file names	`curl -X POST 'https://firewall/api?type=export&category=dlp-pcap&dlp-password=<password>"`
Specific data filtering PCAP file	`curl -X POST 'https://firewall/api?type=export&category=dlp-pcap&dlp-password=<password>&from=<filename>&to=<localfile>"`

Export Certificates and Keys

Use the following procedure to export certificates and keys.

STEP 1 | To export certificates and keys, specify query parameters**certificate-name**,**format**, and**passphrase**:

```
https://<firewall>/api/?
key=apikey&type=export&category=<certificate>
       &certificate-name=<certificate_name>
       &passphrase=<passphrase>
       &format=<pkcs12><pem><pkcs10>
       &include-key=<yes><no>&vsys=<vsys>
       <omit this parameter to import it into a shared location>
```


- **certificate-name**—name of the certificate object on the firewall
- **passphrase**—required when including the certificate key
- **format**—certificate format:**pkcs12**,**pem**, or**pkcs10**
- **include-key**—yes or no parameter to include or exclude the key
- **vsys**—virtual system where the certificate object is used. Ignore this parameter if the certificate is a shared object.

You can use the example above to export a certificate signing request (CSR). If you do so, then specify the following two parameters as shown:

- **format**—**pkcs10**
- **include-key**—**no**

STEP 2 | Confirm that the XML response includes the certificate:

```
-----BEGIN CERTIFICATE-----
MIIDXTCCAkWgAwIBAgIJAJC1HiIAZAiIMA0GCSqGSIb3Df
BAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVx
aWRnaXRzIFB0eSBMdGQwHhcNMTExMjMxMDg1OTQ0WhcNMT
<!-- TRUNCATED -->
-----END CERTIFICATE-----
```

Export Technical Support Data

Debug log data sizes are large, so the API uses an asynchronous job scheduling approach to retrieve technical support data. Learn more about Asynchronous and Synchronous Requests to the PAN-OS XML API. The values for the action parameter are:

- **action=<null>**—When an action parameter is not specified, the system creates a new job to retrieve tech support data. The initial query creates a job ID that you can then use to check on the status of the job, retrieve results, or delete the job.
- **action=status**—Check the status of the job. This returns an XML response with a status element; when the status text data is FIN the job is completed and the tech support file can be retrieved. Example:

```
curl -X POST 'https://firewall/api?type=export&category=tech-
support&action=status&job-id=299"
```

- **action=get**—Retrieve the tech support file as an attachment. The response contains a `application/octet-stream` content-type and a content-disposition header with a suggested filename; for example:

```
Content-Type: application/octet-stream
```



```
    Content-Length: 19658186

    Content-Description: File Transfer

    Content-Transfer-Encoding: binary

    Content-Disposition: attachment; filename=techsupport-8469.tgz
```

- **action=finish**—Stop an active job.

STEP 1 | Create a job to retrieve technical support data.

Use the following request:

```
curl -X POST 'https://firewall/api?type=export&category=tech-
support"
```

The response includes a job ID:

```
<response status="success"code="19"> <result> <msg> <line>Exec job
enqueued with jobid 2</line> </msg> <job>2</job> </result></response>
```

STEP 2 | Check on the status of the job.

Use the job ID returned in the previous response as the job-id parameter:

```
curl -X POST 'https://firewall/api?type=export&category=tech-
support&action=status&job-id=id"
```

A status value of FIN indicates the data is ready to be retrieved.

```
    <response status="success">
  <result>
    <job>
      <tenq>2012/06/14 10:11:09</tenq>
      <id>2</id>
      <user/>
      <type>Exec</type>
      <status>FIN</status>
      <stoppable>no</stoppable>
      <result>0K</result>
      <tfin>10:12:39</tfin>
      <progress>10:12:39</progress>
      <details/>
      <warnings/>
      <resultfile>//tmp/techsupport.tgz</resultfile>
    </job>
  </result>
</response>
```


STEP 3 | Retrieve the tech support data.

```
curl -X POST 'https://firewall/api?type=export&category=tech-
support&action=get&job-id=id"
```

When using cURL, you can specify the output file name as an option to cURL (**-o**). After a successful retrieval of the job data, the job is automatically deleted by the system.

STEP 4 | (Optional) Stop the active job in case of error.

If there is an error or issue with the export job, it may not complete. In cases like this, stop the active job:

```
curl -X POST 'https://firewall/api?type=export&category=tech-
support&action=finish&job-id=id"
```

The response includes a success message:

```
<response status"success">
   <msg>Job  2  removed.</msg>
 </response>
```


Import Files (API)

You can import certain types of files, including as software, content, licenses, and configurations into the firewall using the**type=import** parameter in the API request.

Use**type=import** and specify the category to import these types of files:

- Software—**category=software**
- Content—**category=<anti-virus | content | url-database | signed-url-database>**
- Licenses—**category=license**
- Configuration—**category=configuration**
- Certificates/key—**category=<certificate | high-availability-key | key-pair>**
- Response pages—**category=<application-block-page | captive-portal-text | file-block-continue-page | file-block-page | global-protect-portal-custom-help-page | global-protect-portal-custom-login-page | global-protect-portal-custom-welcome-page | ssl-cert-status-page | ssl-optout-text | url-block-page | url-coach-text | virus-block-page>**
- Clients—**category=global-protect-client**
- Custom logo—**category=custom-logo**
- Importing Basics
- Import Files

Importing Basics

Use cURL to import files to the firewall.

- Import files to a firewall:

```
curl --form file=@
       <filename> “https://firewall/api/?key=apikey&t<query-
parameters>”
```

- Import files to a firewall via Panorama. First import the file to Panorama, then run a request batch upload-install op command:

```
curl -X GET "http://<panorama>/api/?
key=apikey&type=op&cmd=<request><batch><anti-virus><upload-
install><uploaded-file><your-file-name-here></uploaded-
file><devices><serialnumber></devices></upload-install></anti-
virus></batch></request>"
```


Import Files

Use the API Browser to see a full list of import categories.

- Import a certificate or key by specifying the type of the certificate or key file using the**category** parameter:
  - **category=certificate**
  - **category=keypair**
  - **category=high-availability-key**
- (**category=certificate** or**category=keypair only**) Specify these additional parameters for the certificate file and keypair imports:
  - **certificate-name**—name of the certificate object on the firewall
  - **format**—certificate format,**pkcs12** or**pem**
  - **passphrase**—required when including the certificate key
  - **vsys**—virtual system where the certificate object is used. Ignore this parameter if the certificate is a shared object.

```
curl -X POST 'https://firewall/api?
type=import&category=certificate&certificate-
name=<certificate_name>&format=pkcs12 |
 pem&passphrase=text&vsys=<vsys>"
```

- Import a GlobalProtect response pages using an additional parameter for the security profile in which the page should be imported:

```
profile=profilename
```

- Import custom logos to different locations based on the where parameter:

```
where=<login-screen | main-ui | pdf-report-footer | pdf-report-
header>
```


Retrieve Logs (API)

Retrieve logs from a firewall using the API.

- API Log Retrieval Parameters
- Example: Use the API to Retrieve Traffic Logs

API Log Retrieval Parameters

Specify the log type with additional optional parameters to retrieve logs from a firewall.

Parameter	Description
log-type	The type of logs to retrieve: • **`log-type=traffic`**—Traffic logs • **`log-type=threat`**—Threat logs • **`log-type=config`**—Config logs • **`log-type=system`**—System logs • **`log-type=hipmatch`**— GlobalProtect Host Information Profile (HIP) matching logs • **`log-type=globalprotect`**— GlobalProtect logs • **`log-type=wildfire`**—WildFire logs • **`log-type=url`**—URL filtering logs • **`log-type=data`**—Data filtering logs • **`log-type=corr`**—Correlated event logs as seen in the user interface within **Monitor** > **Automated Correlated Engine** > **Correlated Events**. • **`log-type=corr-detail`**—Correlated event details as seen in the user interface when you select an event within **Monitor** > **Automated Correlated Engine** > **Correlated Events**. • **`log-type=corr-categ`**—Correlated events by category, currently compromised hosts seen within **ACC** > **Threat Activity** > **Compromised Hosts**. • **`log-type=userid`**—User-ID logs • **`log-type=auth`**—Authentication logs • **`log-type=gtp`**—GPRS Tunneling Protocol (GTP) logs • **`log-type=external`**—External logs • **`log-type=iptag`**—IP tag logs • **`log-type=decryption`** — Decryption logs

Parameter	Description
query	(Optional) Specify the match criteria for the logs. This is similar to the query provided in the web interface under the Monitor tab when viewing the logs. The query must be URL encoded.
nlogs	(Optional) Specify the number of logs to retrieve. The default is 20 when the parameter is not specified. The maximum is 5000.
skip	(Optional) Specify the number of logs to skip when doing a log retrieval. The default is 0. This is useful when retrieving logs in batches where you can skip the previously retrieved logs.
dir	(Optional) Specify whether logs are shown oldest first (**forward**) or newest first (**backward**). Default is**backward**.
action	(Optional) Log data sizes can be large so the API uses an asynchronous job scheduling approach to retrieve log data. The initial query returns a Job ID (**job-id**) that you can then use for future queries with the**action** parameter: • **action=get**—Check status of an active job or retrieve the log data when the status is**FIN** (finished). This is slightly different than the asynchronous approach to retrieve tech support data where a separate status action is available. • **action=finish**—Stop an active job. • **Not specified**—When not specified, such as during an initial query, the system creates a new job to retrieve log data. *Learn more about* Asynchronous and Synchronous Requests to the PAN-OS XML API.

Example: Use the API to Retrieve Traffic Logs

Follow these steps to use the API retrieve traffic logs.

STEP 1 | Create a job to retrieve all traffic logs that occurred after a certain time:

```
curl -X POST 'https://firewall/api?type=log&log-
type=traffic&query=(receive_time geq '2012/06/22 08:00:00')"
```

A web-browser will automatically URL encode the parameters, but when using cURL, the query parameter must be URL encoded.

Response:


```
      <response status="success" code="19">
  <result>
    <msg>
      <line>query job enqueued with jobid 18</line>
    </msg>
    <job>18</job>
  </result>
</response>
```

STEP 2 | Retrieve traffic log data using the following request using the job ID as the value returned in the previous response:

```
curl -X POST 'https://firewall/api?type=log&action=get&job-id=<id>"
```

STEP 3 | Confirm that the XML response looks similar to the following:

```
      <response status="success"">
       <result>
       <job>...</job>
       <log>
       <logs count="20" progress="100n>
       <entry logid="5753304543500710425"> <domain>1</
domain> <receive_time>2012/06/13 15:43:17</receive_time>
 <serial>001606000117</serial> <segno>6784588</segno>
       <actionflags>0x0</actionflags> <type>TRAFFIC</
type> <subtype>start</subtype> <config_ver>1</config_ver>
 <time_generated>2012/06/13 15:43:17</time_generated>
       <src>172.16.1.2</src> <dst>10.2.0.246</dst>
 <natsrc>10.26.0.96</natsrc> <natdst>10.2.0.246</natdst>
 <rule>default allow</rule>
```

When the job status is FIN (finished), the response automatically includes all the logs in the XML data response. The <log> node in XML is not present when the job status is still pending. After successful log data retrieval, the system automatically deletes the job.

STEP 4 | (Optional) Delete and active log retrieval job.To delete an active log retrieval job, run the following query:

```
curl -X POST 'https://firewall/api?type=log&action=finish&job-
id=<id>"
```

A successful completion returns a job ID.

Apply User-ID Mapping and Populate Dynamic Groups (API)

Use the **type=user-id** parameter to apply User-ID mapping information directly to the firewall. If you are using a third-party VPN solution or have users who are connecting to an 802.1x enabled wireless network, the User-ID API enables you to map users to groups so that you can capture login events and send them to the User-ID agent or directly to the firewall. Additionally, you can use the API to register the IP-to-user mapping information from the input file to populate the members of a dynamic address group or dynamic user group on the firewall.

```
curl -F key=<apikey> --form file=@<filename> "https://<firewall>/
api/?type=user-id"
```

or

```
curl --data-urlencode key=<apikey> -d type=user-id --data-urlencode
 "cmd=xml-document" https://<firewall>/api/
```

With your User-ID API requests, you can use the following optional parameters:

- **vsys=vsys_id**—Specify the vsys where you want to apply User-ID mapping.
- **target=serialnumber**—Specify the firewall by serial number when redirecting through Panorama.

- *Use a GET request if the URL query size is less than 2K and a POST request if the request size is between 2K to 5MB. Limit the query size to 5MB.*
- *When multiple login or logout events are generated at the same time, make sure to follow these guidelines to ensure optimal firewall performance:*
  - *Design your application to queue events and perform batch API updates instead of sending single event or mapping updates.*
  - *Limit the number of concurrent API calls to five. The suggested limit ensures that there is no performance impact to the firewall web interface as the management plane web server handles requests from both the API and the web interface. Limits may vary depending on the type of request. The limit may be higher depending on requests.*
- *(Panorama managed firewalls only) You cannot view the IP addresses of a DAG registered using XML API on the Panorama web interface. You must log in to the Panorama CLI to view the registered IP addresses of a DAG populated using XML API on Panorama and use the following command:*

  ```
  show object registered-ip all
  ```

Use the information in the following table to apply User-ID mapping information to a firewall:

Mapping or Registration Action	API Request

User-ID mapping for a login, logout, or groups.	Use this input file format when providing a User-ID mapping for a login event, logout event, or for groups:
	
	`<uid-message>`
	`<version>1.0</version>`
	`<type>update</type>`
	`<payload>`
	`<login>`
	`<entry name="domain\uid1" ip="10.2.1.1" timeout="20">`
	`</entry>`
	`</login>`
	`<groups>`
	`<entry name="group1">`
	`<members>`
	`<entry name="user1"/>`
	`<entry name="user2"/>`
	`</members>`
	`</entry>`
	`<entry name="group2">`
	`<members>`
	`<entry name="user3"/>`
	`</members>`
	`</entry>`
	`</groups>`
	`</payload>`
	`</uid-message>`
	
	You can include a HIP report by including a **<hip-report></hip-report>** XML container within an **<entry>** parent element.

Multi-User System Entry	Use the following input file format to set up a terminal server entry on the firewall and to specify the port range and block size of ports that will be assigned per user. If you are using the default port range (1025 to 65534) and block size (200) you do not need to send a **multiusersystem setup** message; the firewall will automatically create the terminal server object when it receives the first login message.
	
	`<uid-message>`
	`<payload>`
	`<multiusersystem>`
	`<entry ip="10.2.1.2" startport="xxxxx" endport="xxxxx" blocksize="xxx">`
	`</multiusersystem>`

Mapping or Registration Action	API Request

	<pre></payload>
	<type>update</type>
	<version>1.0</version>
	</uid-message></pre>

User-ID XML multiuser system login event	When the terminal servers sends a login event payload to the firewall, it can contain multiple login events. The firewall uses the information in the information in the login message to populate its user mapping table. For example, if the firewall received a packet with a source address and port of 10.2.1.23:20101, it would map the request to user jparker for policy enforcement.
	<pre><uid-message>
	<payload>
	<login>
	<entry name="acme\jparker"
	ip="10.2.1.23" blockstart="20100">
	</login>
	</payload>
	<type>update</type>
	<version>1.0</version>
	</uid-message></pre>

User-ID XML multiuser system logout	Upon receipt of a logout event message with a**blockstart** parameter, the firewall removes the corresponding IP address-port-user mapping. If the logout message contains a username and IP address, but no**blockstart** parameter, the firewall removes all mappings for the user. If the logout message contains an IP address only, the firewall removes the multi-user system and all associated mappings.
	<pre><uid-message>
	<payload>
	<logout>
	<entry user="domain\uid2"
	ip="10.2.1.2" blockstart="xxxxx">
	</logout>
	</payload>
	<type>update</type>
	<version>1.0</version>
	</uid-message></pre>

Dynamic address group IP address registration	Use the following input file format to dynamically register and unregister IP addresses. You can configure a timeout as part of the member element to automatically unregister IP address-to-tag mapping after a specified amount of time. By default, no timeout is specified meaning the mapping will not timeout and must be manually

Mapping or Registration Action	API Request

	unregistered. Additionally, a timeout of zero (0) seconds does not timeout. You can specify a timeout between zero (0) seconds and 2,592,000 seconds (30 days).
	
	`<uid-message>`
	`<version>1.0</version>`
	`<type>update</type>`
	`<payload>`
	`<register>`
	`<entry ip="10.2.1.1">`
	`<tag>`
	`<member timeout="3600">CBB09C3D-3416-4734-BE90-0395B7598DE3</member>`
	`</tag>`
	`</entry>`
	`</register>`
	`<unregister>`
	`<entry ip="10.2.1.3"/>`
	`<tag>`
	`<member>CBB09C3D-3416-4734-BE90-0395B7598DE5</member>`
	`</tag>`
	`</entry>`
	`</unregister>`
	`</payload>`
	`</uid-message>`

Register tags for a user to add that user to a dynamic user group	Use the following input file format to dynamically register tags to a user and include that user in a dynamic user group.
	
	To register a tag for a user, that user must have an existing user mapping or group mapping.
	
	You can configure a timeout to automatically unregister the user-to-tag mapping after a specified amount of time. By default, no timeout is specified meaning the mapping will not timeout and must be manually unregistered. You can specify a timeout between zero (0) and 2,562,000 seconds (30 days).
	
	`<uid-message>`
	`<version>1.0</version>`
	`<type>update</type>`
	`<payload>`
	`<register-user>`
	`<entry user="paloaltonetworks\john">`
	`<tag>`
	`<member>finished_ethics_training</member>`
	`<member>mac_user</member>`
	`</tag>`

Mapping or Registration Action	API Request
	<pre> </entry> <entry user="paloaltonetworks\jane"> <tag> <member timeout="120">building_1</ member> <member>pc_user</member> </tag> </entry> </register-user> </payload> </uid-message></pre>

Unregister specific tags for a user to remove that user from the dynamic user group	Use the following input file format to unregister a specific dynamic tag from a user and remove the user from the dynamic user group associated with that tag.
	<pre><uid-message>
	  <version>1.0</version>
	  <type>update</type>
	  <payload>
	    <unregister-user>
	      <entry user="paloaltonetworks\john">
	        <tag>
	          <member>mac_user</member>
	        </tag>
	      </entry>
	    </unregister-user>
	  </payload>
	</uid-message></pre>

Unregister all tags for a user	Use the following input file format to unregister all tags for a specific user.
	<pre><uid-message>
	  <version>1.0</version>
	  <type>update</type>
	  <payload>
	    <unregister-user>
	      <entry user="paloaltonetworks\john">
	      </entry>
	    </unregister-user>
	  </payload>
	</uid-message></pre>

Clear all tags for all users	Use the following input file format to unregister all tags from all users.
	<pre><uid-message>
	  <version>1.0</version>
	  <type>update</type></pre>

Mapping or Registration Action	API Request
	<payload> <clear> <registered-user> <all/> </registered-user> </clear> </payload> </uid-message>

Get Version Info (API)

Use the**type=version** request type to show the PAN-OS version for a firewall or Panorama. In addition to the PAN-OS version, this request provides a direct way to obtain the serial number and model number.

STEP 1 | Make a request to the PAN-OS XML API and with**type=version** along with your API key:

```
curl -X POST 'https://firewall/api?type=version&key=<apikey>"
```

STEP 2 | Confirm that the XML response contains the software version, model, serial number, and whether multi-vsys mode is on:

```
     <response status="success">
  <result>
    <sw-version>7.1.0</sw-version>
    <multi-vsys>off</multi-vsys>
    <model>pa-vm</model>
    <serial>007000001222</serial>
  </result>
</response>
```


Get Started with the PAN-OS REST API

To use the PAN-OS® and Panorama™ REST API, first use your administrative credentials to get an API key. You can then use the API key to make API requests.

- PAN-OS REST API
- Access the PAN-OS REST API
- Resource Methods and Query Parameters (REST API)
- PAN-OS REST API Request and Response Structure
- PAN-OS REST API Error Codes
- Work With Objects (REST API)
- Create a Security Policy Rule (REST API)
- Work with Policy Rules on Panorama (REST API)
- Create a Tag (REST API)
- Configure a Security Zone (REST API)
- Configure a Virtual SD-WAN Interface (REST API)
- Create an SD-WAN Policy Pre Rule (REST API)
- Configure an Ethernet Interface (REST API)
- Update a Virtual Router (REST API)
- Work With Decryption (APIs)

The PAN-OS REST API covers a subset of the firewall and Panorama functions, and you'll need to use the XML API to complete the configuration and commit your changes.

The API requests in this guide use cURL commands. However, you can make API requests with other tools such as Postman or a RESTClient. By default, PAN-OS uses a self-signed certificate, so you will need to use the -k parameter with cURL requests. Alternatively, you can replace the self-signed certificate with one from a trusted certificate authority. If you have an internal certificate authority, generate your own certificate and install it on the firewall.

PAN-OS REST API

The PAN-OS® and Panorama™ REST API allow you to manage firewalls and Panorama through a third-party service, application, or script.

You can use the REST API to Create, Read, Update, Delete (CRUD) Objects and Policies on the firewalls; you can access the REST API directly on the firewall or use Panorama to perform these operation on policies and objects from a central location and push them to the managed firewalls.

The inputs in the PAN-OS REST API generally match the web interface, and you can use the PAN-OS Web Interface Help to familiarize yourself with the field properties, descriptions, and supported values for each product. Reading relevant portions of the PAN-OS Administrator's Guide will help you get a better understanding of firewall capabilities that you can access using the API. To use the API, you should also be knowledgeable about web service APIs and HTTP.

For performance considerations, limit the number of concurrent API calls to five. The suggested limit ensures that there is no performance impact to the firewall web interface as the management plane web server handles requests from both the API and the web interface. Limits may vary depending on the type of request. The limit may be higher depending on requests.

To get started, see:

- Access the PAN-OS REST API
- Resource Methods and Query Parameters (REST API)
- PAN-OS REST API Request and Response Structure
- PAN-OS REST API Error Codes
- Work With Objects (REST API)
- Create a Security Policy Rule (REST API)
- Work with Policy Rules on Panorama (REST API)
- Create a Tag (REST API)
- Configure a Security Zone (REST API)
- Configure a Virtual SD-WAN Interface (REST API)
- Create an SD-WAN Policy Pre Rule (REST API)

Access the PAN-OS REST API

The PAN-OS REST API URL format includes a base path and the URI for the endpoint.

`curl -X GET "https://<IP address or FQDN>/restapi/<PAN-OS version>/<resource URI>"`, where:

The base path includes the FQDN or IP address of the firewall or Panorama and the version. The resource URI is the path for the resource or endpoint you want to work with, and it corresponds with the resources you can access on the web interface. Use the PAN-OS Web Interface Help to familiarize yourself with the field properties, descriptions, and supported values for each resource.

You can view the full list of resources in the REST API Reference on the firewall or Panorama at**`https://<IP_address>/restapi-doc .`**

To use the REST API, you must Enable API Access for your administrators and Get Your API Key. See API Authentication and Security for details on authenticating your API requests.

The following table lists the PAN-OS 11.0 REST API resource URIs that are available on the firewall. The resource URIs on Panorama are analogous except that resources support both pre rule and post rule policies. The PAN-OS 11.0 REST API resources offer abilities like managing policies on the firewall or configuring SD-WAN interfaces and policies on Panorama. To complete the configuration, you'll need to use the XML API on the firewall and Panorama.

Resource	URI
OBJECTS	/restapi/v11.0/Objects/Addresses
	/restapi/v11.0/Objects/AddressGroups
	/restapi/v11.0/Objects/Regions
	/restapi/v11.0/Objects/DynamicUserGroups
	/restapi/v11.0/Objects/Applications
	/restapi/v11.0/Objects/ApplicationGroups
	/restapi/v11.0/Objects/ApplicationFilters
	/restapi/v11.0/Objects/Services
	/restapi/v11.0/Objects/ServiceGroups
	/restapi/v11.0/Objects/Tags
	/restapi/v11.0/Objects/GlobalProtectHIPObjects

Resource	URI
	/restapi/v11.0/Objects/GlobalProtectHIPProfiles
	/restapi/v11.0/Objects/ExternalDynamicLists
	/restapi/v11.0/Objects/CustomDataPatterns
	/restapi/v11.0/Objects/CustomSpywareSignatures
	/restapi/v11.0/Objects/CustomVulnerabilitySignatures
	/restapi/v11.0/Objects/CustomURLCategories
	/restapi/v11.0/Objects/AntivirusSecurityProfiles
	/restapi/v11.0/Objects/AntiSpywareSecurityProfiles
	/restapi/v11.0/Objects/VulnerabilityProtectionSecurityProfiles
	/restapi/v11.0/Objects/URLFilteringSecurityProfiles
	/restapi/v11.0/Objects/FileBlockingSecurityProfiles
	/restapi/v11.0/Objects/WildFireAnalysisSecurityProfiles
	/restapi/v11.0/Objects/DataFilteringSecurityProfiles
	/restapi/v11.0/Objects/DoSProtectionSecurityProfiles
	/restapi/v11.0/Objects/SecurityProfileGroups
	/restapi/v11.0/Objects/LogForwardingProfiles
	/restapi/v11.0/Objects/AuthenticationEnforcements

Resource	URI
	/restapi/v11.0/Objects/DecryptionProfiles
	/restapi/v11.0/Objects/PacketBrokerProfiles
	/restapi/v11.0/Objects/SDWANPathQualityProfiles
	/restapi/v11.0/Objects/SDWANTrafficDistributionProfiles
/restapi/v11.0/Objects/SDWANSaasQualityProfiles	
/restapi/v11.0/Objects/SDWANErrorCorrection	
/restapi/v11.0/Objects/Schedules	
POLICIES	/restapi/v11.0/Policies/SecurityRules
	/restapi/v11.0/Policies/NATRules
	/restapi/v11.0/Policies/QoSRules
	/restapi/v11.0/Policies/PolicyBasedForwardingRules
	/restapi/v11.0/Policies/DecryptionRules
	/restapi/v11.0/Policies/NetworkPacketBrokerRules
	/restapi/v11.0/Policies/TunnelInspectionRules
	/restapi/v11.0/Policies/ApplicationOverrideRules
	/restapi/v11.0/Policies/AuthenticationRules
	/restapi/v11.0/Policies/DoSRules
/restapi/v11.0/Policies/SDWANRules	
NETWORK	/restapi/v11.0/Network/EthernetInterfaces

Resource	URI
	/restapi/v11.0/Network/AggregateEthernetInterfaces
	/restapi/v11.0/Network/VLANInterfaces
	/restapi/v11.0/Network/LoopbackInterfaces
	/restapi/v11.0/Network/TunnelIntefaces
	/restapi/v11.0/Network/SDWANInterfaces
	/restapi/v11.0/Network/Zones
	/restapi/v11.0/Network/VLANs
	/restapi/v11.0/Network/VirtualWires
	/restapi/v11.0/Network/VirtualRouters
	/restapi/v11.0/Network/IPSecTunnels
	/restapi/v11.0/Network/GRETunnels
	/restapi/v11.0/Network/DHCPServers
	/restapi/v11.0/Network/DHCPRelays
	/restapi/v11.0/Network/DNSProxies
	/restapi/v11.0/Network/GlobalProtectPortals
	/restapi/v11.0/Network/GlobalProtectGateways
	/restapi/v11.0/Network/GlobalProtectGatewayAgentTunnels
	/restapi/v11.0/Network/GlobalProtectGatewaySatelliteTunnels
	/restapi/v11.0/Network/GlobalProtectGatewayMDMServers
	/restapi/v11.0/Network/GlobalProtectClientlessApps

Resource	URI
	/restapi/v11.0/Network/GlobalProtectClientlessAppGroups
	/restapi/v11.0/Network/QoSInterfaces
	/restapi/v11.0/Network/LLDP
	/restapi/v11.0/Network/GlobalProtectIPSecCryptoNetworkProfiles
	/restapi/v11.0/Network/IKEGatewayNetworkProfiles
	/restapi/v11.0/Network/IKECryptoNetworkProfiles
	/restapi/v11.0/Network/MonitorNetworkProfiles
	/restapi/v11.0/Network/InterfaceManagementNetworkProfiles
	/restapi/v11.0/Network/ZoneProtectionNetworkProfiles
	/restapi/v11.0/Network/QoSNetworkProfiles
	/restapi/v11.0/Network/LLDPNetworkProfiles
	/restapi/v11.0/Network/BFDNetworkProfiles
/restapi/v11.0/Network/SDWANInterfaceProfiles	
Devices	/restapi/v11.0/Device/VirtualSystems
/restapi/v11.0/Network/SNMPTrapServerProfiles	
/restapi/v11.0/Network/SyslogServerProfiles	
/restapi/v11.0/Network/EmailServerProfiles	
/restapi/v11.0/Network/HttpServerProfiles	
/restapi/v11.0/Network/LDAPServerProfiles	

Resource Methods and Query Parameters (REST API)

The PAN-OS REST API requires query parameters for all API requests. The following table describes the methods that the PAN-OS REST API supports and includes the query parameters required for each operation.

For a list of all resource URIs, see Access the PAN-OS REST API. To start using the API, see Work With Objects (REST API) or Create a Security Policy Rule (REST API).

Resource Method		Read the list of resources	Create a resource	Modify a resource	Delete a resource	Rename a resource	Move a policy rule(Policies only)
HTTP Method		**GET**	**POST**	**PUT**	**DELETE**	**POST**	**POST**
Query Parameters	name	optional	required	required	required	required	required

	location	required,
		valid values on the firewall:
		`predefin`
		,
		`shared` for Objects only
		,
		`vsys`
		, or
		`panorama pushed`
			required,
			valid values on the firewall:
			`shared` for Objects only
			,
			`vsys`
			valid values on Panorama:
			`shared`
			or
			`device-group`
				required,
				valid values on the firewall:
				`shared` for Objects only
				,
				`vsys`
				valid values on Panorama:
				`shared`
				or
				`device-group`
					required,
					valid values on the firewall:
					`shared` for Objects only
					,
					`vsys`
					valid values on Panorama:
					`shared`
					or
					`device-group`
						required,
						valid values on the firewall:
						`shared` for Objects only
						,
						`vsys`
						valid values on Panorama:
						`shared`
						or
						`device-group`
							required,
							valid values on the firewall:
							`shared`
							,
							`vsys`
							valid values on Panorama:
							`shared`
							or
							`device-group`

	vsys	required, if	required, if	required, if	required, if	required, if	required, if

	location is `vsys` or `panorama-pushed`	location is `vsys`	location is `vsys`	location is `vsys`	location is `vsys`	location is `vsys`
device-group	(Panorama only) required, if location is device-group	(Panorama only) required, if location is device-group	(Panorama only) required, if location is device-group	(Panorama only) required, if location is device-group	(Panorama only) required, if location is device-group	(Panorama only) required, if location is device-group
input-format	—	optional, default format is JSON	optional, default format is JSON	—	—	—
output-format	optional, default format is JSON	optional, default format is JSON	optional, default format is JSON	optional, default format is JSON	optional, default format is JSON	optional, default format is JSON
newname	—	—	—	—	required	—
where	—	—	—	—	—	required, valid values: `top`, `bottom`, `before`, `after`

	dst	—	—	—	—	—	required, when `where` is `before` or `after`
Request Body		—	required	required	—	—	—

The following table shows examples of request formats with query parameters.

Action	Example of Query Parameters in URL
List	`GET https://<firewall or Panorama IP>/restapi/v11.0/<resource URI>?location=location&output-format=json`
Create	`POST https://<firewall or Panorama IP>/restapi/v11.0/<resource URI>?location=location&name=name`
Edit	`PUT https://<firewall or Panorama IP>/restapi/v11.0/<resource URI>?location=location&name=name`
Delete	`DELETE https://<firewall or Panorama IP>/restapi/v11.0/<resource URI>?location=location&name=name`
Rename	`POST https://<firewall or Panorama IP>/restapi/v11.0/<resource URI>:rename?location=location&name=name &newname=newname`
Move	`POST https://<firewall or Panorama IP>/restapi/v11.0/<resource URI>:move?location=location&name=name &where=<move to>` <move to> can be**top**,**bottom**,**before &dst=<policy name>**, after &dst=<policy name>

PAN-OS REST API Request and Response Structure

The PAN-OS REST API enables you to perform CRUD operations with objects and use them in policy rules. A resource in the PAN-OS REST API is an endpoint that you can configure with parameters. When you make requests with the endpoints, you get responses that contain information. The request and response formats support JSON (default) and XML.

After you configure the firewalls and Panorama using the REST API, you must use the XML API or the other management interfaces to commit your changes to the running configuration.

Request Format

The API request format is constructed as shown in the example below:

```
curl -X GET "https://<IP address or FQDN of the firewall or
 Panorama>/restapi/<PAN-OS version>/<resource URI>?<query
 parameters>request body"
```

- Base path and the resource URI for the endpoint. See Access the PAN-OS REST API for details.
- Query parameters. Every request includes query parameters that are passed to the API endpoint using query strings. The query parameters are appended to the URL with a ? that indicates the start of the query string. The query parameters appear after the ?, the parameter are concatenated with other parameters using the ampersand & symbol.

Query Parameters on the firewall	• name (name) of the resource. • location (vsys, predefined, shared, panorama-pushed) of the resource on which you want to perform the operation. A predefined object or rule is built-in to the firewall and you cannot edit, rename or delete predefined objects or policy rules. • virtual system (vsys) name for the resource, if location is vsys or panorama-pushed. • input format (input-format). JSON is default, or XML. You can specify an input format for HTTP methods that have a request body, such as PUT to update and POST to create a resource. • output format (output-format) JSON default, or XML
Query Parameters on Panorama	• name (name) of the resource. • location (predefined, shared, device-group, panorama-pushed) of the resource on which you want to perform the operation.

	A predefined object or rule is built-in to Panorama and you cannot edit, rename or delete predefined objects or policy rules. • device group (device-group) name of the Panorama device group to which you have assigned the firewalls, if location is device-group. • input format (input-format). JSON is default, or XML. You can specify an input format for HTTP methods that have a request body, such as PUT to update and POST to create a resource. • output format (output-format) JSON default, or XML

- Request body. When you create a resource with a POST request or edit a resource with a PUT request, you include a JSON or XML formatted request body in which you specify the properties for the resource you want to create or modify on the endpoint.

When you make an API request to the firewall or Panorama, the API key is required to authenticate the user who is making the request. You can enter the key with the custom HTTP header ***X-PAN-KEY: <key>****. Learn about* API Authentication and Security *and how to* Get Your API Key.

Success Response Format

The HTTP response for a successful call has three elements: status, code, and result. The code is a numeric value. Refer to the PAN-OS XML API Error Codes for details on the code included in the HTTP response message.


```
  {
"@code": "19",
"@status": "success",
"result": {
    "@count": "3",
    "@total-count": "3",
    "entry": [
        {
            "@location": "vsys",
            "@name": "fqdn1",
            "@vsys": "vsys1",
            "fqdn": "www.test.com"
        },
        {
            "@location": "vsys",
            "@name": "Peer1",
            "@vsys": "vsys1",
            "ip-netmask": "172.0.0.1/24"
        },
        {
            "@location": "vsys",
```



```
                "@name": "Peer2renamed",
                "@oldname": "Peer2",
                "@vsys": "vsys1",
                "ip-netmask": "200.0.0.1/24"
            }
        ]
    }
}
```

Error Response Format

In addition to the HTTP status code, the error response includes a JSON object or XML with error information. The following is an example of an error response body from a REST API call to get an address. In this example, the request is missing a query parameter:

```
  {
    "code": 3,
    "details": [
        {
            "@type": "CauseInfo",
            "causes": [
                {
                    "code": 7,
                    "description": "Missing Query Parameter: name",
                    "module": "panui_restapi"
                }
            ]
        }
    ],
    "message": "Missing Query Parameter: name"
}
```

The following table describes the error response fields.

Field	Description
code	Feature-specific error code. The codes are listed in PAN-OS REST API Error Codes
message	Human-readable message that corresponds to the code
details	Array of objects containing detailed data about the error
details.@type	Type of data in details. Currently, the only type available is CauseInfo.

Field	Description
	The details under CauseInfo are for readability and debugging purposes. The value can change between software releases. To avoid your scripts breaking between releases, don't parse the values in details.causes.
details.causes	Array of objects that convey module-level error data
details.causes.module	Feature-specific module that reported the error
details.causes.code	Module-level error code. If details.causes.module is panui_mgmt, then you can find this module-level code in PAN-OS REST API Error Codes
details.causes.description	Details about the error, from the feature-specific module.

PAN-OS REST API Error Codes

The possible REST API feature-specific error response codes and their descriptions are as follows:

Error Code	Description
1	The operation was canceled, typically by the caller.
2	Unknown internal server error.
3	Bad request. The caller specified an invalid parameter.
4	Gateway timeout. A firewall or Panorama module timed out before a backend operation completed.
5	Not found. The requested entity was not found.
6	Conflict. The entity that the caller attempted to create already exists.
7	Forbidden. The caller does not have permission to execute the specified operation.
16	Unauthorized. The request does not have valid authentication credentials to perform the operation.
8	Resource exhausted. Some resource has been exhausted.
9	Failed precondition. The operation was rejected because the system is not in a state required for the execution of the operation.
10	Aborted because of conflict. A typical cause is a concurrency issue.
11	Out of range. The operation was attempted past a valid range. And example is reaching an end-of-file.
12	Not implemented. The operation is disabled, not implemented, or not supported.
13	Internal server error. An unexpected and potentially serious internal error occurred.
14	Service unavailable. The service is temporarily unavailable.
15	Internal server error. Unrecoverable data loss or data corruption occurred.

Work With Objects (REST API)

Objects are elements that you use within policy rules. The firewalls and Panorama support a large number of objects such as tags, address objects, log forwarding profiles, and security profiles.

The examples in this section show you how to perform CRUD operations with an address object. You can use this example to work with other objects of the firewall. Access the REST API reference documentation at**https://<IP address or FQDN of the firewall or Panorama>/restapi-doc/**for help with the resource URIs for different objects and the structure of the request. For an overview, see PAN-OS REST API Request and Response Structure.

- Create an Address Object
- Edit an Address Object
- Rename an Address Object
- Delete an Address Object
- Get Address Objects

Create an Address Object

Make a POST request to create an address object. In the request, the query parameters must include the name and the location on where you want to create the object. And in the request body include the same name, location and other properties to define the object. For example:

```
curl -X POST \
     'https://10.2.1.4/restapi/v11.0/Objects/Addresses?
location=shared&name=web-servers-production' \
     -H 'X-PAN-KEY: LUFRPT0=' \
     -d '{
    "entry": [
        {
            "@location": "shared",
            "@name": "web-servers-production",
            "description": "what is this for?",
            "fqdn": "docs.paloaltonetworks.com",
            "tag": {
                "member": [
                    "blue"
                ]
            }
        }
    ]
}'
```

Edit an Address Object

Make a PUT request and include the name and location of the object as query parameters. Include the same location and name in the request body and define the properties of the object you'd like to change. In the following example, you are modifying the description and adding a new tag

called red to the address object. If the tag does not already exist, you must first create the tag before you can reference it in the address object.

```
curl -X PUT \
     'https://10.2.1.4/restapi/v11.0/Objects/Addresses?
location=shared&name=web-servers-production' \
     -H 'X-PAN-KEY: LUFRPT0=' \
     -d '{
    "entry": [
        {
            "@location": "shared",
            "@name": "web-servers-production",
            "description": "publish servers",
            "fqdn": "docs.paloaltonetworks.com",
            "tag": {
                "member": [
                    "blue",
                    "red"
                ]
            }
        }
    ]
}'
```

The response is

```
    {
    "@code": "20",
    "@status": "success",
    "msg": "command succeeded"
}
```

Rename an Address Object

When renaming an object, make a POST request with the following query parameters—name of the object**name=<name>**, l**ocation=<location>**, and the new name**newname=<name>**. The following example renames web-servers-production to web-server-publish.

```
curl -X POST \
     'https://10.5.196.4/restapi/v11.0/Objects/Addresses:rename?
location=shared&name=web-servers-production&newname=web-server-
publish' \
     -H 'X-PAN-KEY: LUFRPT0='
```

Delete an Address Object

Make a DELETE request and include the name and the location of the object as query parameters. For example:

```
curl -X DELETE \
```



```
     'https://10.2.1.4/restapi/v11.0/Objects/Addresses?
location=shared&name=web-server-production' \
     -H 'X-PAN-KEY: LUFRPT0='
```

Get Address Objects

Make a GET request to retrieve a list of all address objects within a specified location. For example, the following query reads all address objects in vsys1 which is indicated with**location=vsys&vsys=vsys1**in the query parameter.

```
curl -X GET \
     'https://10.2.1.4/restapi/v11.0/Objects/Addresses?
location=vsys&vsys=vsys1' \
     -H 'X-PAN-KEY: LUFRPT0='
```

And the response includes the list of address objects that are configured on vsys1 on the firewall.

```
    {
    "@code": "19",
    "@status": "success",
    "result": {
        "@count": "3",
        "@total-count": "3",
        "entry": [
            {
                "@location": "vsys",
                "@name": "fqdn1",
                "@vsys": "vsys1",
                "fqdn": "www.test.com"
            },
            {
                "@location": "vsys",
                "@name": "Peer1",
                "@vsys": "vsys1",
                "ip-netmask": "172.0.0.1/24"
            },
            {
                "@location": "vsys",
                "@name": "Peer2renamed",
                "@oldname": "Peer2",
                "@vsys": "vsys1",
                "ip-netmask": "200.0.0.1/24"
            }
        ]
    }
}
```


Create a Security Policy Rule (REST API)

The example in this section shows you how to create and update a Security policy rule on the firewall. Use this example to get familiar with the REST API and then make it work with other policy types on the firewall. Access the REST API reference documentation at**https://<IP address or FQDN of the firewall or Panorama>/restapi-doc/**for help with the resource URIs for the different objects and policies and for help with the properties supported for each type of request. For an overview, see PAN-OS REST API Request and Response Structure.

- Create an Application Object
- Create a Security Policy Rule
- Reference an Address Object in the Rule

Create an Application Object

Make a POST request to create an application object that allows you to allow browser-based applications that belong to the category collaboration and subcategory email. To make this application object named**email-collaboration-apps** available across all virtual systems on a firewall, create the object at**location=shared**. Use Palo Alto Networks Applipedia, the application database to view the attributes (Category, Subcategory, Technology, Risk or Characteristic) that you can use to define the object. You can also refer to**https://<firewall_IP>/restapi-doc/#tag/objects-applications** for details on how to construct an application object. Here is an example.

```
curl -X POST \
  'https://10.2.1.4/restapi/v11.0/Objects/Applications?
location=shared&name=email-collaboration-apps' \
  -H 'X-PAN-KEY: LUFRPT=' \
  -d '{
    "entry": [
        {
            "@location": "shared",
            "@name": "email-collaboration-apps",
            "able-to-transfer-file": "yes",
            "category": "collaboration",
            "description": "apps we allow for collaboration",
            "risk": "2",
            "subcategory": "email",
            "technology": "browser-based"
        }
    ]
}'
```

You can now use this application object in a Security policy rule.

Create a Security Policy Rule

Before you start here, use the XML API or any of the other management interfaces to set up interfaces and zones on the firewall.

To create a Security policy rule, make a POST request. In the following example, the API key is provided as a custom header X-PAN-KEY instead of as query parameter. For more details, see

Access the PAN-OS REST API. The query parameters include the name of the rule, location and vsys name**location=vsys&vsys=<vsys_name>&name=<rule_name>**. And in the request body specify the same name, location, vsys name, and includes additional properties for the Security policy rule including the application object you created earlier.

```
curl -X POST \

  'https://10.2.1.4/restapi/v11.0/Policies/SecurityRules?
location=vsys&vsys=vsys1&name=rule-example1' \
  -H 'X-PAN-KEY: LUFRPT=' \
  -d '{
    "entry": [
        {
            "@location": "vsys",
            "@name": "rule-example1",
            "@vsys": "vsys1",
            "action": "allow",
            "application": {
                "member": [
                    "email-collaboration-apps"
                ]
            },
            "category": {
                "member": [
                    "any"
                ]
            },
            "destination": {
                "member": [
                    "any"
                ]
            },
            "from": {
                "member": [
                    "zone-edge1"
                ]
            },
            "source-hip": {
             "member": [
              "any"
             ]
            },
            "destination-hip": {
             "member": [
              "any"
             ]
            },
            "service": {
                "member": [
                    "application-default"
                ]
            },
            "source": {
                "member": [
                    "any"
                ]
```



```
            },
            "source-user": {
                "member": [
                    "any"
                ]
            },
            "to": {
                "member": [
                    "any"
                ]
            }
        }
    ]
}'
```

Instead of using an application object, you can list applications by name as long as the applications are included in the application content version installed on the firewall.

```
 "application": {
    "member": [
        "gmail",
        "linkedin",
        "sendgrid",
        "front"
    ]
}
```

Reference an Address Object in the Rule

To allow access to only specific addresses in the source zone, you can include an address object and restrict access to only those members in the source zone with **"source": {"member": ["web-servers-production"]}** as shown in the following example:

```
curl -X PUT \
  'https://10.2.1.4/restapi/v11.0/Policies/SecurityRules?
location=vsys&name=rule-example1&vsys=vsys1' \
  -H 'X-PAN-KEY: LUFRPT=' \
  -d '{
    "entry": [
        {
            "@location": "vsys",
            "@name": "rule-example1",
            "@vsys": "vsys1",
            "action": "allow",
            "application": {
                "member": [
                    "email-collaboration-apps"
                ]
            },
            "category": {
                "member": [
                    "any"
                ]
            },
            "destination": {
```

```
                "member": [
                    "any"
                ]
            },
            "from": {
                "member": [
                    "zone-edge1"
                ]
            },
            "source-hip": {
             "member": [
              "any"
             ]
            },
            "destination-hip": {
             "member": [
              "any"
             ]
            },
            "service": {
                "member": [
                    "application-default"
                ]
            },
            "source": {
                "member": [
                    "web-servers-production"
                ]
            },
            "source-user": {
                "member": [
                    "any"
                ]
            },
            "to": {
                "member": [
                    "any"
                ]
            }
        }
    ]
}'
```

If successful, the response is

```
{"@status": "success","@code": "20","msg":"command succeeded"
    }
}
```

If the address object does not exist, the response is as follows:

```
{"code": 3,"message": "Invalid Object","details": [
        {"@type": "CauseInfo","causes": [
                {"code": 12,"module": "panui_mgmt","description":
 "Invalid Object:  rule-example1 -> source 'web-servers-production'
 is not an allowed keyword.  rule-example1 -> source web-servers-
```



```
production is an invalid ipv4/v6 address. rule-example1 -> source
 web-servers-production invalid range start IP.
 rule-example1 -> source 'web-servers-production' is not a valid
 reference. rule-example1 -> source is invalid."
                }
            ]
        }
    ]
}
```


Work with Policy Rules on Panorama (REST API)

On Panorama, you create policy rules as Pre Rules or Post Rules and then push them from Panorama to the managed firewalls. While you can view these rules on the managed firewalls, you can edit the Pre Rules and Post Rules only on Panorama. Pre Rules are added to the top of the rule order and are evaluated first, and Post Rules are added after any locally defined rules on the firewall and are at the bottom of the rule hierarchy, so they evaluated last. Post Rules typically include rules to deny access to traffic based on the App-ID, User-ID, or Service. Pre Rules and Post Rules are of two types: Shared Post Rules are shared across all managed devices and device groups, and device group Post Rules are specific to a device group.

The example in this section shows you how to create and update a Security policy rule on Panorama. Use this example to get familiar with the REST API and then make it work with other policy types on the firewall. Access the REST API reference documentation at**https://<Panorama IP address or FQDN>/restapi-doc/**for help with the resource URIs for the different objects and policies and for help with the properties supported for each type of request. For an overview, see PAN-OS REST API Request and Response Structure.

- Create a Log Forwarding Profile
- Edit a Security Policy Pre Rule

Create a Log Forwarding Object

Make a POST request to create an log forwarding object that allows you to forward traffic and threat logs to the Logging Service. To make this log forwarding object named**log-forwarding-LS** available for all firewalls in the device group named**devicegroup-7**, create the object at**location=devicegroup-7**. Include the name of the object, specify the location as device-group and the device-group name in the query parameters**location=device-group&device-group=<dg_name>&name=<object_name>**and create the request body. The API key is provided as a custom header X-PAN-KEY.

```
curl -X POST \
     'http://10.5.1.70/restapi/v11.0/Objects/LogForwardingProfiles?
name=log-forwarding-LS&location=device-group&device-
group=devicegroup-7' \
     -H 'X-PAN-KEY: LUFRPT1=' \
     -d ' {
    "entry": {
        "@name": "log-forwarding-LS",
        "match-list": {
            "entry": [
                {
                    "@name": "only_traffic_logs",
                    "filter": "All Logs",
                    "log-type": "traffic",
                    "send-to-panorama": "yes"
                },
                {
                    "@name": "only_threat_logs",
                    "filter": "All Logs",
                    "log-type": "threat",
                    "send-to-panorama": "yes"
```



```
                }
            ]
        }
    }
}'
```

You can now use this log forwarding object in a Security policy rule.

Edit a Security Policy Pre Rule

To modify a Security policy Pre Rule, make a PUT request to**https://<Panorama IP address or FQDN>/restapi/v11.0/Policies/SecurityPreRules**. The query parameters include the name of the rule, location and device group name if the location is a device group**location=device-group&device-group=<device_group_name>&name=<rule_name>**. And in the request body specify the same name, location, device group name, and include the required properties for the Security policy pre rule. This example shows you how to reference the log forwarding object you created earlier. Refer to the REST API reference documentation at**https://<Panorama IP address or FQDN>/restapi-doc/** for help with the required and optional properties in the request body.

Use a GET request to fetch the configuration of the Security policy pre rule you want to modify and copy the response. You can then use this as a starting point for the request body in your PUT request and modify as needed to edit the rule.

```
curl -X PUT \
      'http://10.2.1.7/restapi/v11.0/Policies/SecurityPreRules?
LOCATION=device-group&device-group=devicegroup-7&name=allow-dns' \
      -H 'X-PAN-KEY: LUFRPT=' \
      -d '{
    "entry": [
        {
            "@device-group": "devicegroup-7",
            "@location": "device-group",
            "@name": "allow-dns",
            "action": "allow",
            "application": {
                "member": [
                    "dns"
                ]
            },
            "category": {
                "member": [
                    "any"
                ]
            },
            "destination": {
                "member": [
                    "any"
                ]
            },
            "from": {
                "member": [
```



```
                    "any"
                ]
            },
            "source-hip": {
             "member": [
              "any"
             ]
            },
            "destination-hip": {
             "member": [
              "any"
             ]
            },
            "log-setting": "log-forwarding-LS",
            "log-start": "yes",
            "service": {
                "member": [
                    "application-default"
                ]
            },
            "source": {
                "member": [
                    "any"
                ]
            },
            "source-user": {
                "member": [
                    "any"
                ]
            },
            "target": {
                "negate": "no"
            },
            "to": {
                "member": [
                    "any"
                ]
            }
        }
    ]
}
```


The response body indicates the success or failure of the request. If you reference a Security policy Pre Rule that does not exist because the name of the rule is invalid or the location is incorrect, the response displays as


```
     {
    "code": 5,
    "details": [
        {
            "@type": "CauseInfo",
            "causes": [
                {
                    "code": 7,
```



```
                    "description": "Object Not Present: No object to
 edit.",
                    "module": "panui_mgmt"
                }
            ]
        }
    ],
    "message": "Object Not Present"
}
```

For help with the error codes, see PAN-OS REST API Error Codes.

Create a Tag (REST API)

Tags allow you to group objects using keywords or phrases.

Link tags are tags that enable you use to identify groups of physical interfaces specifically for an SD-WAN configuration on Panorama™. Some examples of link tags are Low Cost Paths, General Access, Private HQ, and Backup. The following is an example of a REST API request to create a link tag.

```
curl -X POST 'https://<Panorama>/restapi/v11.0/objects/tags?
location=device-group&device-group=SD-WAN_Branch&name=Low-Cost-Paths'
    -H 'X-PAN-KEY: <your key>
    -d '{
      "entry": {
         "@name": "Low-Cost-Paths”,
         “comments”: “Groups two low cost broadband links and a
 backup link”
         }
     }'
```


Configure a Security Zone (REST API)

Security zones are a logical way to group physical and virtual interfaces on the firewall to control and log the traffic that traverses specific interfaces on your network. You must assign an interface on the firewall to a security zone before that interface can process traffic. A zone can have multiple interfaces of the same type, but an interface can belong to only one zone.

Create a Security Zone

You can create a security zone either directly on the firewall or as part of a network template on Panorama™.

Make a REST API request to add a security zone.

The following example shows you how to use a Panorama REST API request to create a security zone with Ethernet interfaces and a virtual SD-WAN interface. See Configure an SD-WAN Interface (REST API) for an example of a REST API request to create a virtual SD-WAN interface through Panorama and Configure an Ethernet Interface (REST API) for an example of a REST API request on the firewall to configure an Ethernet interface.

```
        curl -X POST
        'https://<Panorama>/restapi/v11.0/network/zones?
location=template&template=SDWAN-Branch-Network&name=Untrust'
        -H 'X-PAN-KEY: <api key>'
        -d '{
    "entry": {
        "@name": "Untrust",
        "enable-user-identification": "no",
        "network": {
            "layer3": {
                "member": [
                    "ethernet1/1",
                    "ethernet1/2",
                    "ethernet1/3",
                    "sdwan.1"
                ]
            }
        }
    }
}'
```

Update a Security Zone

To update a security zone, you should first make a REST API request to get the existing security zone. You can then copy data from the response to your REST API request to update the zone to ensure no desired existing data is inadvertently lost or overwritten. The following example first retrieves an existing security zone from a PAN-OS firewall and then updates the zone by adding a new Ethernet interface.

STEP 1 | Get the zone you to which you want to add the Ethernet interface.

The following example requests an existing security zone.

```
        curl -X GET 'https://<firewall>/restapi/v11.0/network/
zones?name=test&location=vsys&vsys=vsys1' \
        -H 'X-PAN-KEY: <api key>'
```

The response is shown below. Note that this security zone already has on Ethernet interface, ethernet1/4. You need to include that member in your request to update this zone to avoid losing this data.

```
        {
    "@code": "19",
    "@status": "success",
    "result": {
        "@count": "1",
        "@total-count": "1",
        "entry": [
            {
                "@location": "vsys",
                "@name": "test",
                "@vsys": "vsys1",
                "network": {
                    "layer3": {
                        "member": [
                            "ethernet1/4"
                        ]
                    }
                }
            }
        ]
    }
}
```

STEP 2 | Add a new Ethernet interface and include any existing data.

The following example updates the security zone with (1) a new Ethernet interface, ethernet1/3 and (2) the member that already existed in the zone, ethernet1/4.

```
        curl -X PUT 'https://<firewall>/restapi/v11.0/network/
zones?location=vsys&vsys=vsys1&name=test' \
        --header 'X-PAN-KEY: <api key>' \
        -d '{
    "entry": {
        "@name": "test",
        "enable-device-identification": "no",
        "enable-user-identification": "no",
        "network": {
            "layer3": {
                "member": [
```



```
                    "ethernet1/4",
                    "ethernet1/3"
                ]
            }
        }
    }
}'
```

Configure an SD-WAN Interface (REST API)

A virtual SD-WAN interface groups multiple physical links use to communicate with the same destination.

This example shows you how to create a virtual SD-WAN interface on Panorama™. This interface is meant for direct Internet access from a branch, so the SD-WAN interface will include only physical Ethernet interfaces. It's assumed that you've already set up necessary templates and device groups on Panorama. The examples use a template called**SD-Branch-Network**.

STEP 1 | Create a link tag.

A link tag enables you use to group physical links so that SD-WAN path selection and traffic redirection can use the groups to maximize application and service quality. See Create a Tag (REST API) for an example of an API request to create a link tag.

STEP 2 | Create an SD-WAN interface profile.

Create an SD-WAN interface profile to define the characteristics of ISP connections and to control the speed of links and how frequently the firewalls monitors the link. This profile includes both the link tag you created and the type of link that the interface is (ADSL/DLS, Cable modem, Ethernet, Fiber, LTE/3G/4G/5G, MPLS, etc.). The following POST request creates an SD-WAN interface profile.

```
curl -X POST
       'https://<Panorama>/restapi/v9.1/network/
sdwanInterfaceprofiles?location=template&template=SDWAN-Branch-
Network&name=BroadBand-low-cost'
       -H 'X-PAN-KEY: <api key>
       -d '{
    "entry": {
        "@name": "BroadBand-low-cost",
        "comment": "Low cost",
        "failback-hold-time": 20,
        "link-tag": "Broadband-ISP",
        "link-type": "Cablemodem",
        "maximum-download": 100,
        "maximum-upload": 50,
        "path-monitoring": "Aggressive",
        "probe-frequency": 5,
        "probe-idle-time": 60,
        "vpn-data-tunnel-support": "yes"
    }
}'
```

STEP 3 | Configure one or more physical interfaces.

Configure the physical interface(s) that the virtual SD-WAN interface will include. This example configures one Ethernet interface. Configuring an Ethernet interface for SD-WAN involves three steps, which are (a) configure a Layer 3 Ethernet interface without an SD-WAN

interface profile, (b) import the Ethernet interface into a virtual system, and (c) update the Ethernet interface to specify the SD-WAN interface profile.

1. Configure a Layer 3 Ethernet interface.

   The following is an example of an API request to configure a Layer 3 Ethernet interface that uses DHCP for IP address assignment.

```
curl -X POST
         'https://<Panorama>/restapi/v9.1/network/
ethernetinterfaces?location=template&template=SDWAN-Branch-
Network&name=ethernet1/4'
         -H 'X-PAN-KEY: <api key>
         -d '{
    "entry": {
        "@name": "ethernet1/4",
        "layer3": {
            "dhcp-client": {
                "create-default-route": "yes",
                "default-route-metric": 10,
                "enable": "yes",
                "send-hostname": {
                    "enable": "no",
                    "hostname": "system-hostname"
                }
            },
            "sdwan-link-settings": {
                "enable": "no"
            }
        }
    }
}'
```

2. Import the Ethernet interface into a virtual system (vsys).

   Make an API request on Panorama to import the Ethernet interface into a vsys configuration. The example below imports the newly created Ethernet interface into**vsys1**, which exists in template**Branch_template**. In this example, there is only one interface. If other interfaces already exist in the vsys, though, include them all in the interface member list.

```
curl -X POST
         'https://<Panorama>/restapi/v9.1/device/
virtualsystems?location=template&template=SDWAN-Branch-
Network&name=vsys1'
         -H 'X-PAN-KEY: <api key>
         -d '{
    "entry": [
        {
            "@location": "template",
            "@name": "vsys1",
            "@template": "Branch_template",
            "import": {
                "network": {
                    "interface": {
```



```
                        "member": [
                            "ethernet1/4"
                        ]
                    }
                }
            }
        }
    ]
}'
```

3. Apply an SD-WAN interface profile to the Ethernet interface.

   The example below applies an SD-WAN interface profile to the Ethernet interface to complete the Ethernet interface configuration for SD-WAN.

```
curl -X PUT
         'https://<Panorama>/restapi/v9.1/network/
ethernetinterfaces?location=template&template=SDWAN-Branch-
Network&name=ethernet1/4'
         -H 'X-PAN-KEY: <api key>
         -d '{
    "entry": {
        "@name": "ethernet1/4",
        "layer3": {
            "dhcp-client": {
                "create-default-route": "yes",
                "default-route-metric": 10,
                "enable": "yes",
                "send-hostname": {
                    "enable": "no",
                    "hostname": "system-hostname"
                }
            },
            "sdwan-link-settings": {
                "enable": "yes",
                "sdwan-interface-profile": "BroadBand-test"
            }
        }
    }
}'
```

STEP 4 | Configure a virtual SD-WAN interface.

The following is an example to configure a virtual SD-WAN interface.

1. Create a parent SD-WAN interface named **sdwan** if one doesn't already exist.

   The following example creates the parent interface **sdwan** for template **SDWAN-Branch-Network**.

```
curl -X POST
         'https://<Panorama>/restapi/v9.1/network/
sdwanInterfaces?location=template&template=SDWAN-Branch-
Network&name=sdwan'
         -H 'X-PAN-KEY: <api key>
```



```
          -d '{
    "entry": {
        "@name": "sdwan"
    }
}'
```

2. Create and configure an SD-WAN interface.

   Specify one or more SD-WAN-capable Ethernet interfaces that have the same destination, for example directly to the Internet. The following example creates a virtual SD-WAN interface that has two Ethernet interfaces, including the Ethernet interface you configured earlier.

```
curl -X POST
         'https://<Panorama>/restapi/v9.1/network/
sdwanInterfaces?location=template&template=SDWAN-Branch-
Network&name=sdwan.1'
         -H 'X-PAN-KEY: <api key>
         -d '{
    "entry": {
        "@name": "sdwan.1",
        "interface": {
            "member": [
                "ethernet1/3",
                "ethernet1/4"
            ]
        }
    }
}'
```


Create an SD-WAN Policy Pre Rule (REST API)

An SD-WAN policy rule specifies when and how a firewall performs application-based SD-WAN path selection. You can configure an SD-WAN policy pre rule or post rule on Panorama™ and push the rule to the firewalls in your device group.

The examples in this section show how to use the REST API to create an SD-WAN policy pre rule on Panorama. An SD-WAN policy rule includes both a path quality profile and a traffic distribution profile. The policy rule uses these two profiles to identify network quality requirements and to determine path selection when the network doesn't meet those quality requirements.

STEP 1 | Create a path quality profile.

A path quality profile identifies network quality or health requirements based on packet loss percentage, jitter, and latency. Once included in an SD-WAN policy rule, the path quality profile will control the threshold at which the firewall replaces a deteriorating path with a new path for matching application packets. A number of predefined path quality profiles exist, but you can create your own if none of the existing profiles meets your needs. The following POST request creates a path quality profile called**general-business2**.

```
curl -X POST
       'https://<Panorama>/restapi/v9.1/objects/
sdwanpathqualityprofiles?location=device-group&device-group=SD-
WAN_Branch&name=general-business2'
       -H 'X-PAN-KEY: <api key>
       -d '{
    "entry": {
        "@name": "general-business2",
        "metric": {
            "jitter": {
                "sensitivity": "medium",
                "threshold": 20
            },
            "latency": {
                "sensitivity": "medium",
                "threshold": 300
            },
            "pkt-loss": {
                "sensitivity": "medium",
                "threshold": 5
            }
        }
    }
}'
```

STEP 2 | Create a traffic distribution profile.

Create a traffic distribution profile, which specifies how a firewall determines a new best path if the current preferred path exceeds a path quality threshold. A traffic distribution profile specifies one of three possible distribution methods: Best Available Path, Top-Down Priority,

and Weighted Session Distribution. The profile also includes one or more link tags, which the distribution method uses to narrow its selection of a new path.

The POST request below creates a traffic distribution profile that uses top-down priority and includes two link tags:**Broadband-ISP** and**LTE-ISP**. See Create a Tag (REST API) for an example of a REST API request to create a link tag.

```
curl -X POST
       'https://<Panorama>/restapi/v9.1/objects/
sdwantrafficdistributionprofiles?location=device-group&device-
group=SD-WAN_Branch&name=BroadBand2'
       -H 'X-PAN-KEY: <api key>
       -d '{"entry": {"@name": "BroadBand2","traffic-distribution":
 "Top Down Priority","link-tags": {"entry": [
       {"@name": "Broadband-ISP"
       },
       {"@name": "LTE-ISP",
       }
       ]
       }
       }
       }'
```

STEP 3 | Create an SD-WAN policy pre rule.

An SD-WAN policy pre rule specifies application(s) and/or service(s) and a traffic distribution profile to determine how a firewall selects the preferred path for an incoming packet that doesn't belong to an existing session and that matches all other criteria. Examples of the criteria are source and destination zones, source and destination IP addresses, and source user. The SD-WAN policy pre rule also specifies a path quality profile of thresholds for packet loss, jitter, and latency. When one of the thresholds is exceeded, the firewall selects a new path for the application(s) and/or service(s).

The POST request below creates an SD-WAN policy pre rule that Panorama will push to a device group called**SD-WAN_Branch**. The request body parameters include both a path quality profile and a traffic distribution profile. The parameters also include**Trust-PA220** and**Wireless-PA220** as the source zones and**Untrust-PA220** as the destination zone. See Configure a Security Zone (REST API) for an example of a REST API request to create a zone.

```
curl -X POST
       'https://<Panorama>/restapi/v9.1/policies/
sdwanprerules?location=device-group&device-group=SD-
WAN_Branch&name=HQ_Service_Test'
       -H 'X-PAN-KEY: <api key>'
       -d '{"entry": {"@name": "HQ_Service_Test","from": {"member":
 ["Trust-PA220”
       ]
       },"to": {"member": ["Untrust-PA220"
       ]
       },"source": {"member": ["any”
       ]
       },"source-user": {"member": ["any"
       ]
       },"destination": {"member": ["any"
```



```
        ]
        },"application": {"member": ["ping"
        ]
        },"service": {"member": ["any"
        ]
        },"negate-source": "no","negate-destination":
 "no","disabled": "no","description": "For SD-WAN test","path-
quality-profile": "general-business","action": {"traffic-
distribution-profile": "BroadBand2"
        }
        }
        }'
```


Configure an Ethernet Interface (REST API)

There are multiple deployment options for Ethernet interfaces on firewalls. Three common options are: Tap, Virtual Wire, and Layer 3.

The following example shows how to configure a Layer 3 Ethernet interface. Configuration of a Layer 3 Ethernet interface on a firewall involves two REST API requests: (1) A request to configure the interface and (2) a request to import the interface into the virtual system.

The example includes the creation of an interface management profile that you assign to the Layer 3 Ethernet interface. While an interface management profile is optional for configuring the interface, this profile has an important role because it provides protection from unauthorized access.

STEP 1 | Configure an interface management profile (Optional).

An interface management profile protects the firewall from unauthorized access by defining the services and IP addresses that a firewall interface permits. The following example creates an interface management profile that allows only ping and response pages. This example restricts IP addresses that can access the interface to 192.168.1.0/24, but if there are no IP restrictions required, then don't add entries to the**permitted-ip** list.

```
       curl -X POST https://<firewall>/restapi/v11.0/network/
interfacemanagementnetworkprofiles?name=ping-and-response-pages'
       -H 'X-PAN-KEY: <api key>
       -d '{
    "entry": {
        "@name": "ping-and-response-pages",
        "http": "no",
        "http-ocsp": "no",
        "https": "no",
        "permitted-ip": {
            "entry": [
                {
                    "@name": "192.168.1.0/24"
                }
            ]
        },
        "ping": "yes",
        "response-pages": "yes",
        "snmp": "no",
        "ssh": "no",
        "telnet": "no",
        "userid-service": "no",
        "userid-syslog-listener-ssl": "no",
        "userid-syslog-listener-udp": "no"
    }
}'
```


STEP 2 | Configure a Layer 3 Ethernet interface.

The following is an example of an API request to configure a Layer 3 Ethernet interface that uses DHCP for IP address assignment. The configuration includes application of the interface management profile you configured in step 1.

```
       curl -X POST https://<firewall>/restapi/v11.0/network/
ethernetinterfaces?name=ethernet1/3'
       -H 'X-PAN-KEY: <api key>
       -d '{
    "entry": {
        "@name": "ethernet1/3",
        "layer3": {
            "dhcp-client": {
                "create-default-route": "yes",
                "default-route-metric": 10,
                "enable": "yes",
                "send-hostname": {
                    "enable": "no",
                    "hostname": "system-hostname"
                }
            },
            "interface-management-profile": "ping-and-response-
pages"
        }
    }
}'
```

STEP 3 | Import the Ethernet interface into your virtual system (vsys).

The following example updates the import section of the firewall virtual system**vsys1** with the Ethernet interface you configured in step 2.

```
       curl -X POST https://<firewall>/restapi/v11.0/device/
virtualsystems?name=vsys1'
       -H 'X-PAN-KEY: <api key>
       -d '{
    "entry": [
        {
            "@name": "vsys1",
            "import": {
                "network": {
                    "interface": {
                        "member": [
                            "ethernet1/3"
                        ]
                    }
                }
            }
        }
    ]
}'
```


STEP 4 | Add this interface to a security zone.

The steps above complete the configuration of the Ethernet interface, but for the interface to process network traffic, you must add the interface to a security zone. See Update a Security Zone for an example of REST API requests to add an Ethernet interface to an existing security zone.

STEP 5 | Add the Ethernet interface to an existing virtual router, like the default virtual router.

The firewall requires a virtual router to obtain routes to other subnets through either participating L3 routing protocols (dynamic routes) or static routes. See Update a Virtual Router (REST API) for an example of REST API requests to add an interface to a virtual router.

Update a Virtual Router (REST API)

A virtual router allows the firewall to route traffic from one network to another through its Layer 3 interfaces or static routes. Each Layer 3 interface, loopback interface, and VLAN interface defined on the firewall must be associated with a virtual router. Each interface can belong to only one virtual router.

The following steps show how to add an existing Ethernet interface to the predefined virtual router,**`default`**.

STEP 1 | Retrieve the existing virtual router named default.

Before you update an existing virtual router by adding a new interface, you should retrieve the virtual router to identify interfaces that are already assigned to that virtual router.

```
 curl -X GET 'https://<firewall>/restapi/v11.0/Network/
VirtualRouters' --header 'X-PAN-KEY: <api-key>'
```

A successful response returns a list of existing virtual routers and includes detailed information for each one. The response object for this request is large, so the example response below shows a partial response object. You can see the list of existing interfaces, which will be necessary to include as request body parameters if you are updating the interface list in the virtual router. If the router were to include other configuration items such as protocol information or routes, you would need to include these in the update as well.

```
{
  "@status": "success",
  "@code": "19",
  "result": {
    "@count": "1",
    "entry": [
      {
        "@name": "default",
        "@location": "panorama-pushed",
        "interface": {
          "member": [
            "ethernet1/4"
          ],
          "routing-table": {},
          "protocol": {},
          "admin-dists": {},
          "ecmp": {}
        }
      }
    ]
  }
}
```


STEP 2 | Update the existing virtual router named**default**.

Update the virtual router **default** with a new L3 Ethernet interface,**ethernet1/3**, through a PUT request. Include all information that currently exist as well as any new information in the default Virtual Router in your PUT request.

```
curl --location --request PUT 'https://<firewall>/restapi/v11.0/
Network/VirtualRouters?name=default'
--header 'X-PAN-Key:<api-key>
--data '{
    "entry": {
        "@name": "default",
        "interface": {
            "member": [
                "ethernet1/3",
                "ethernet1/4"
            ]
        }
    }
}'
```


Work With Decryption (APIs)

Use the REST API to automate the workflow when you set up decryption policy rules for your firewalls. This example shows how to create a decryption profile and a decryption forwarding profile and then to include both in a decryption policy rule. With decryption policy rules, you can decrypt traffic and send decryption logs to support private analysis where third-party security appliances can add additional enforcement for traffic that the firewall should allow. You must have a Network Packet Broker license for this example. Review How Network Packet Broker Works for more information about decryption forwarding and creating a security chain,.

This example describes setting up a Layer 3 security chain to forward decrypted SSL traffic (see Layer 3 Security Chain Guidelines).

STEP 1 | Configure two Layer 3 interfaces over which to forward decrypted traffic.

This following POST request configures the Ethernet interface `ethernet1/6` with decryption forwarding for use as a dedicate interface for decrypted traffic.

```
curl -X POST 'https://10.55.152.39/restapi/v11.0/Network/
EthernetInterfaces?name=ethernet1/6'
          -H 'X-PAN-KEY: LUFRP=='
          -d '{
    "entry": {
        "@name": "ethernet1/6",
        "layer3": {
            "decrypt-forward": "yes",
            "lldp": {
                "enable": "no"
            },
            "ndp-proxy": {
                "enabled": "no"
            }
        }
    }
}'
```

The resulting success message:

```
      {
    "@code": "20",
    "@status": "success",
    "msg": "command succeeded"
}
```

STEP 2 | Create a virtual router to enable decryption port forwarding.

The following POST requests uses two Ethernet interfaces dedicated to decryption: `ethernet1/5` and `ethernet1/6`. The virtual router must be dedicated to the decryption

forwarding interfaces to ensure that the clear text sessions that the firewall forwards for additional analysis are completely separated from dataplane traffic.

```
curl -X POST 'https://10.55.152.39/restapi/v11.0/Network/
VirtualRouters?name=decrypttest'
          -H 'X-PAN-KEY: LUFRP==’
          -d ' {
    "entry": {
        "@name": "decrypttest",
        "ecmp": {
            "algorithm": {
                "ip-modulo": {}
            }
        },
        "interface": {
            "member": [
                "ethernet1/5",
                "ethernet1/6"
            ]
        },
        "protocol": {
            "bgp": {
                "enable": "no",
                "routing-options": {
                    "graceful-restart": {
                        "enable": "yes"
                    }
                }
            },
            "ospf": {
                "enable": "no"
            },
            "ospfv3": {
                "enable": "no"
            },
            "rip": {
                "enable": "no"
            }
        }
    }
}'
```

The resulting success message:

```
      {
    "@code": "20",
    "@status": "success",
    "msg": "command succeeded"
}
```


STEP 3 | Create a Decryption Profile.

The following POST request creates a decryption profile that defines the traffic and settings for blocking and allowing traffic in a decryption policy rule. For information on each of the options available for configuration, review how to Define Traffic to Decrypt.

```
curl -X POST 'https://10.55.152.39/restapi/v11.0/Objects/
DecyptionProfiles?name=jl-test&location=vsys&=vsys1&input-
format=json'
       -h 'X-PAN-KEY: LUFRPT'
       -d     '{
    "entry": {
        "@name": "decryptProfileTest",
        "ssh-proxy": {
            "block-if-no-resource": "no",
            "block-ssh-errors": "no",
            "block-unsupported-alg": "no",
            "block-unsupported-version": "no"
        },
        "ssl-forward-proxy": {
            "auto-include-altname": "no",
            "block-client-cert": "no",
            "block-expired-certificate": "no",
            "block-if-no-resource": "no",
            "block-timeout-cert": "no",
            "block-tls13-downgrade-no-resource": "no",
            "block-unknown-cert": "no",
            "block-unsupported-cipher": "no",
            "block-unsupported-version": "no",
            "block-untrusted-issuer": "no",
            "restrict-cert-exts": "no",
            "strip-alpn": "no"
        },
        "ssl-inbound-proxy": {
            "block-if-no-resource": "no",
            "block-tls13-downgrade-no-resource": "no",
            "block-unsupported-cipher": "no",
            "block-unsupported-version": "no"
        },
        "ssl-no-proxy": {
            "block-expired-certificate": "no",
            "block-untrusted-issuer": "no"
        },
        "ssl-protocol-settings": {
            "auth-algo-md5": "no",
            "auth-algo-sha1": "yes",
            "auth-algo-sha256": "yes",
            "auth-algo-sha384": "yes",
            "enc-algo-3des": "yes",
            "enc-algo-aes-128-cbc": "yes",
            "enc-algo-aes-128-gcm": "yes",
            "enc-algo-aes-256-cbc": "yes",
            "enc-algo-aes-256-gcm": "yes",
            "enc-algo-chacha20-poly1305": "yes",
            "enc-algo-rc4": "yes",
            "keyxchg-algo-dhe": "yes",
```



```
            "keyxchg-algo-ecdhe": "yes",
            "keyxchg-algo-rsa": "yes",
            "max-version": "tls1-2",
            "min-version": "tls1-0"
        }
    }
}'
```

The resulting success message:

```
      {
    "@code": "20",
    "@status": "success",
    "msg": "command succeeded"
}
```

STEP 4 | Create a Decryption Forwarding Profile.

The following POST request creates a bidirectional security chain with devices at 1.1.1.1 and 1.1.1.2 using the Ethernet interfaces you created earlier in this task.

```
curl -X POST 'https://10.55.152.39/restapi/
v11.0/Objects/DecryptionForwardingProfiles?
name=decryptionForwardTest&location=vsys&vsys=vsys1'
         -H 'X-PAN-KEY: LUFRP=='
         -d '{
    "entry": {
        "@location": "vsys",
        "@name": "decryptionForwardTest",
        "@vsys": "vsys1",
        "flow": "bidirectional",
        "health-check": {
            "http-enable": "no",
            "http-latency-enable": "no",
            "path-enable": "no"
        },
        "interface-primary": "ethernet1/5",
        "interface-secondary": "ethernet1/6",
        "routed": {
            "security-chain": {
                "entry": [
                    {
                        "@name": "testchain",
                        "enable": "yes",
                        "first-device": "1.1.1.1",
                        "last-device": "1.1.1.2"
                    }
                ]
            }
        }
    }
}'
```


The resulting success message:

```
      {
    "@code": "20",
    "@status": "success",
    "msg": "command succeeded"
}
```

STEP 5 | Create a decryption policy using the decryption profile and decryption forwarding profile you created before.

The following POST requests defines the traffic source zones and destinations to enable decryption based on the `testdecryptionprofile` and `testdecryptionforwading` profiles.

```
curl -X POST 'https://10.55.152.39/restapi/v11.0/Policies/
DecryptionRules?name=jltestrule&location=vsys&vsys=vsys1'
         -H 'X-PAN-KEY: LUFRP'
         -d '{
    "entry": {
        "@location": "vsys",
        "@name": "jltestrule",
        "@uuid": "b4d66137-9678-4b9d-9105-e881899d1125",
        "@vsys": "vsys1",
        "action": "decrypt-and-forward",
        "category": {
            "member": [
                "any"
            ]
        },
        "destination": {
            "member": [
                "any"
            ]
        },
        "destination-hip": {
            "member": [
                "any"
            ]
        },
        "forwarding-profile": "testdecryptionforwarding",
        "from": {
            "member": [
                "l3-untrust"
            ]
        },
        "negate-source": "no",
        "profile": "testdecryptionprofile",
        "service": {
            "member": [
                "any"
```



```
            ]
        },
        "source": {
            "member": [
                "Test"
            ]
        },
        "source-hip": {
            "member": [
                "any"
            ]
        },
        "source-user": {
            "member": [
                "any"
            ]
        },
        "to": {
            "member": [
                "l2-trust"
            ]
        },
        "type": {
            "ssl-forward-proxy": {}
        }
    }
}'
```

The resulting success message:

```
      {
    "@code": "20",
    "@status": "success",
    "msg": "command succeeded"
}
```