Cisco Secure Firewall Threat Defense REST API Guide

First Published: 2018-03-29

Last Modified: 2023-12-13

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

CONTENTS

CHAPTER 1

About the Secure Firewall Threat Defense REST API

You can use the Secure Firewall Threat Defense REpresentational State Transfer (REST) Application Programming Interface (API), over HTTPS, to interact with the threat defense device through a client program. The REST API uses JavaScript Object Notation (JSON) format to represent objects.

Secure Firewall device manager includes an API Explorer that explains all of the resources and JSON objects available for your programmatic use. The Explorer provides detailed information about the attribute-value pairs in each object, and you can experiment with the various HTTP methods to ensure you understand the coding required to use each resource. The API Explorer also provides examples of the URLs required for each resource.

You can also find reference information and examples online at https://developer.cisco.com/site/ftd-api-reference/.

The API is has its own version number. There is no guarantee that a client designed for one version of the API will work for a future version without error or without requiring changes to your program.

Audience for This Programming Guide

This guide is written on the assumption that you have a general knowledge of programming and a specific understanding of REST APIs and JSON. If you are new to these technologies, please first read a general guide on REST APIs.

Supported HTTP Methods

You can use the following HTTP methods only. The other methods are not supported.

- GET—To read data from the system.

- POST—To create new objects.
- PUT—To modify existing objects. When using PUT, you must include the entire JSON object. You cannot selectively update individual attributes within an object.
- DELETE—To remove a user-defined object.

The Base URL for the API

The easiest way to determine the base URL for a given threat defense device is to try out a GET method in the API Explorer, and simply delete the object part of the URL from the result.

For example, you can do a GET /object/networks, and see something similar to the following in the returned output under Request URL:

```
https://ftd.example.com/api/fdm/v1/object/networks
```

The server name part of the URL is the hostname or IP address of the threat defense device, and will be different for your device in place of "ftd.example.com." In this example, you delete /object/networks from the path to get the base URL:

```
https://ftd.example.com/api/fdm/v1/
```

All resource calls use this URL as the base for the request URL.

If you changed the HTTPS data port, you must include the custom port in the URL. For example, if you changed the port to 4443: https://ftd.example.com:4443/api/fdm/v1/

The "v" element in the URL is the API version, and this will typically change with the software version. For example, the API version for the threat defense version 6.3.0 is v2, so the base URL would be:

```
https://ftd.example.com/api/fdm/v2/
```

Note Starting with the threat defense 6.4, you can avoid the need to update the path in your API calls by using **latest** instead of the v element in the path. For example, https://ftd.example.com/api/fdm/latest/. The **latest** alias resolves to the most recent API version supported by the device.

In the API Explorer, if you scroll to the bottom of the page, you can see information on the base URL (without the server name) and API version.

Securing SSL/TLS Communications for the REST API

The Threat Defense devices come with a self-signed certificate so that you can initiate HTTPS communications with the device. However, because the certificate is not signed by a known Certificate Authority (CA), any SSL/TLS access attempt will consider the connection insecure.

When connecting with a browser, you are prompted to accept the self-signed certificate, but a command such as curl will reject the certificate. In the case of curl, you can bypass the certificate check failure by adding the **--insecure** keyword. For example:

```
curl --insecure -X GET --header 'Accept: application/json'
'https://ftd.example.com/api/versions'
```

One of the first things you should do is obtain a CA-signed device certificate for the threat defense device. Then, using the device manager or the API, assign this certificate as the management certificate. Subsequently, SSL/TLS certificate checking should not fail, and you will not need to use insecure communications in your API calls.

Procedure

Step 1 Upload the CA-signed device certificate using the **POST /object/internalcertificates** resource.

Step 2 Make this certificate the management certificate using the **PUT /devicesettings/default/webuicertificates/{objId}** resource.

Use the **GET /devicesettings/default/webuicertificates** resource to determine the object ID for the web UI certificate.

Step 3 Deploy the changes using the **POST /operational/deploy** resource.

Determining the Supported API Versions

You can determine which API versions are supported on a device using the GET /api/versions (ApiVersions) method. This method does not require authentication, and it also does not include a version element in the path. For example:

```
curl -X GET --header 'Accept: application/json' 'https://ftd.example.com/api/versions'
```

Replace “ftd.example.com” with the hostname or IP address of the threat defense device.

This method returns a list of API versions you can use. For example:

```
{
    "supportedVersions":["v3", "latest"]
}
```

The version strings are the same ones you use in the URL for subsequent API calls. If you use **latest** instead of the specific version identifier, you can avoid the need to update your calls for subsequent releases. However, using this technique does not overcome changes to the object models used in your calls, which might need adjustment from release to release.

Typically, your next step would be to get an access token, as described in Authenticating Your REST API Client Using OAuth, on page 13.

API Version Backward Compatibility

The threat defense API version changes with each major release of the threat defense software. New features impact the API calls for the features being added or changed.

However, many features do not change from release to release. For example, the APIs related to network and port objects often remain unchanged in a new release.

Starting with the threat defense version 6.7, if an API resource model for a feature does not change between releases, then the threat defense API can accept calls that are based on the older API version. Even if the feature model did change, if there is a logical way to convert the old model to the new model, the older call can work. For example, a v5 call can be accepted on a v6 system. If you use "latest" as the version number in your calls, these "older" calls are interpreted as a v6 call in this scenario, so whether you are taking advantage of backward compatibility depends on how you are structuring your API calls.

If a feature model has changed between API versions in a way that backward compatibility cannot be supported, you will get an error message, and you will need to check for these errors and update your code for these specific calls.

CHAPTER 2

The API Explorer

Use the API Explorer to learn about the REST API. You can also test the various methods and resources to verify you are configuring them correctly. You can copy and paste the JSON models into your code as a starting point.

Tip The purpose of the API Explorer is to help you learn the API. Testing calls through the API Explorer requires the creation of access locks that might interfere with regular operation. We recommend that you use the API Explorer on a non-production device.

Opening the API Explorer

The API Explorer explains all of the resources and JSON objects available for your programmatic use. The Explorer provides detailed information about the attribute-value pairs in each object, and you can experiment with the various HTTP methods to ensure you understand the coding required to use each resource.

Procedure

Step 1 Using a browser, open the home page of the system, for example, https://ftd.example.com.

Step 2 Log into the device manager.

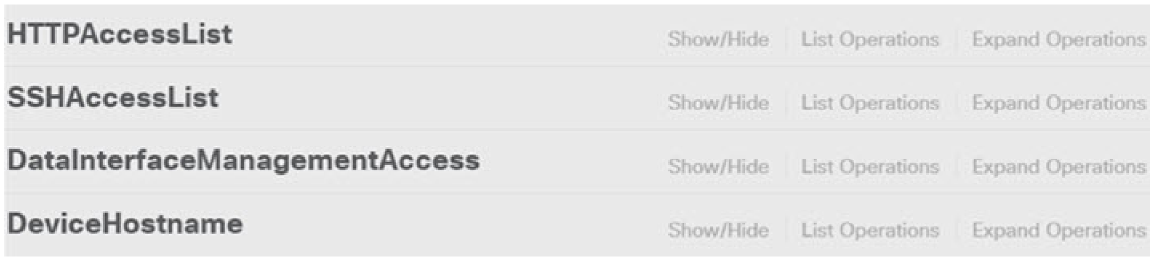
Step 3 (6.4 and earlier.) Edit the URL to point to /#/api-explorer, for example, https://ftd.example.com/#/api-explorer.

Step 4 (6.5 and later.) Click the more options button (⋮) and choose **API Explorer**.

The system opens the API Explorer in a separate tab or window, depending on your browser settings.

Finding Your Way Around API Explorer

When you enter the API Explorer, you are presented with a list of resource groups. These groups include the resources available in the API. The following illustration shows a small sample of the list.



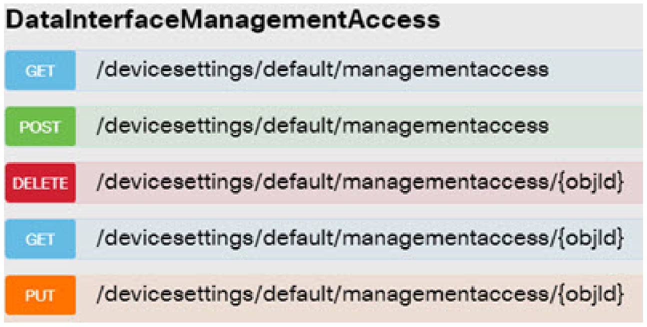
HTTPAccessList	Show/Hide	List Operations	Expand Operations
SSHAccessList	Show/Hide	List Operations	Expand Operations
DataInterfaceManagementAccess	Show/Hide	List Operations	Expand Operations
DeviceHostname	Show/Hide	List Operations	Expand Operations

These group names are links: click the link to open the group to see the methods you can use with the resources in the group. Each group also includes the following commands on the right:

- **Show/Hide** opens and closes the group. This is the same as clicking the group name. Initially, the expansion simply shows the methods (same as **List Operations**), but the system remembers the last expanded state (before you closed it) and re-opens to the same degree of expansion.
- **List Operations** shows the HTTP methods available for each resource in the group. The information includes the relative path for the universal resource locator (URL) template for each resource. Path variables are indicated by the standard convention: {*variable*}. You need to replace {*variable*}, including the braces, with an appropriate value. You must add the base URL to this relative path; see The Base URL for the API, on page 2.

  Click an operation URL template to see complete documentation for that method.
- **Expand Operations** opens all of the HTTP methods and resources available for in a group.

Some groups have many children resources. For example, the DataInterface ManagementAccess group includes GET, POST, and DELETE operations for /devicesettings/default/managementaccess, and GET and PUT operations for /devicesettings/default/managementaccess/{objId}.



DataInterfaceManagementAccess

GET	/devicesettings/default/managementaccess
POST	/devicesettings/default/managementaccess
DELETE	/devicesettings/default/managementaccess/{objId}
GET	/devicesettings/default/managementaccess/{objId}
PUT	/devicesettings/default/managementaccess/{objId}

Viewing Documentation About a Resource

The attributes in each resource are documented in the API Explorer.

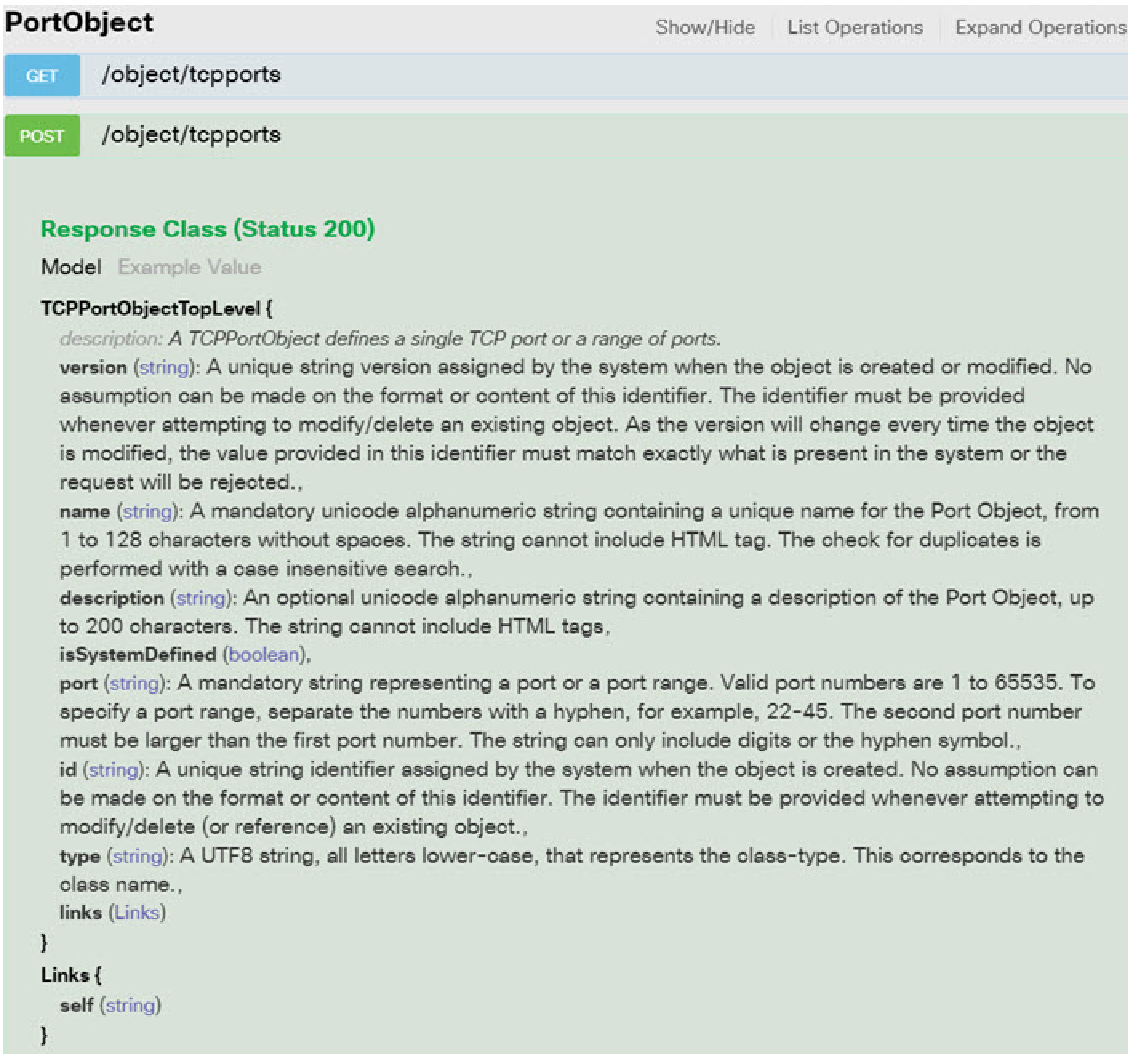
Procedure

Step 1 Drill down to the specific resource and method that interests you.

Step 2 In the **Response Class** section, click the **Model** tab.

The model lists the attributes with descriptions and data types. For GET, there are also paging options that might be returned. If there are more objects than were returned in the response, you get the URLs for the next and previous batch of objects.

For example, the following graphic shows the POST /object/tcpports method and resource, with the **Model** tab selected. By default, the **Example Value** tab is selected, so you must always click **Model** to see the documentation.



```
TCPPortObjectTopLevel {
  description: A TCPPortObject defines a single TCP port or a range of ports.
  version (string): A unique string version assigned by the system when the object is created or modified. No
  assumption can be made on the format or content of this identifier. The identifier must be provided
  whenever attempting to modify/delete an existing object. As the version will change every time the object
  is modified, the value provided in this identifier must match exactly what is present in the system or the
  request will be rejected.,
  name (string): A mandatory unicode alphanumeric string containing a unique name for the Port Object, from
  1 to 128 characters without spaces. The string cannot include HTML tag. The check for duplicates is
  performed with a case insensitive search.,
  description (string): An optional unicode alphanumeric string containing a description of the Port Object, up
  to 200 characters. The string cannot include HTML tags,
  isSystemDefined (boolean),
  port (string): A mandatory string representing a port or a port range. Valid port numbers are 1 to 65535. To
  specify a port range, separate the numbers with a hyphen, for example, 22-45. The second port number
  must be larger than the first port number. The string can only include digits or the hyphen symbol.,
  id (string): A unique string identifier assigned by the system when the object is created. No assumption can
  be made on the format or content of this identifier. The identifier must be provided whenever attempting to
  modify/delete (or reference) an existing object.,
  type (string): A UTF8 string, all letters lower-case, that represents the class-type. This corresponds to the
  class name.,
  links (Links)
}
Links {
  self (string)
}
```

Finding the Object ID (objId) and Parent ID

Some resources require an object ID or related parent object ID in the URL, such as the following:

- PUT /object/networks/{objId}
- GET /policy/intrusionpolicies/{parentId}/intrusionrules

In most cases, you can get the object or parent ID by using a GET method one level higher in the resource hierarchy. The object/parent ID is the UUID on the **id** parameter for a given object.

For example, GET /object/networks returns a list of all network objects currently defined. You might need to do multiple calls to page through the list to get to the object you want, or include the **limit** query parameter to increase the number of objects returned for a call. Each object has the following format; the object ID is highlighted.

```
{
      "version": "9bbb9e5d-8115-11e7-8cb4-772d7eb1894d",
      "name": "any-ipv4",
      "description": null,
      "subType": "NETWORK",
      "value": "0.0.0.0/0",
      "isSystemDefined": true,
      "id": "9bbbc56e-8115-11e7-8cb4-01865c95f930",
      "type": "networkobject",
      "links": {
        "self": "https://ftd.example.com/api/fdm/latest/object/networks/
9bbbc56e-8115-11e7-8cb4-01865c95f930"
      }
```

Note In some cases, the {objId} occurs at the top-level of a hierarchy. In these cases, sometimes you can enter any value for object ID and get the same results. In other cases, examine the object model documentation for information on valid object types; the ID is one of the valid types. These are always GET calls. For example, GET /operational/systeminfo/{objId} and GET /operational/featureinfo/{objId}.

Viewing the Error Catalog And Evaluating Error Messages

As a REST API, the system returns standard HTTP error codes, such as 404 if you perform a GET for an object that does not exist.

In addition, the system includes a number of error messages that explain errors more specifically. If your API call results in an error, the response body might include these more specific messages.

For example, if you try to **POST /object/networks** the following network object, you will get an error. In this case, you are trying to specify a network, but you forgot to include a netmask (that is, the value should be either 10.10.10.0/24 or 10.10.10.0/255.255.255.0):

```
{
  "name": "test-network",
  "subType": "NETWORK",
  "value": "10.10.10.0",
  "type": "networkobject"
}
```

The result would be an HTTP response code of 422, and a response body that includes a specific error message:

```
{
  "error": {
    "severity": "ERROR",
    "key": "Validation",
    "messages": [
      {
        "description": "The type Network requires a netmask. To specify a single host,
either use the type Host, or use {0}/255.255.255.255.",
        "code": "networkWithoutNetmask",
        "location": "value"
      }
    ]
  }
}
```

The following procedure explains how to view the list of possible error messages that can be returned in a response body.

Procedure

Step 1 In the device manager, click the more options button (⋮) and choose **API Explorer**.

Step 2 Click **Error Catalog** in the table of contents.

The messages have the following components.

- **Category**—The general type of message. This value appears in the **key** attribute of the error response body. Categories include Validation, General, and Deployment.
- **Code**—A unique string that identifies the error message. This value appears in the **code** attribute of the error response body. You can use the browser's Find On Page feature to locate messages in the catalog using this value.
- **Message**—The specific message that explains the error. This value appears in the **description** attribute of the error response body. Variables within the message are indicated as {0}, {1}, and so forth.

CHAPTER 3

General Process for Using the REST API

General Process for Using the REST API

In general, your client interacts with the threat defense device using the following iterative process:

1. Obtain an access token to authenticate your API calls. See Overview of the API Client Authentication Process, on page 13.
2. Except when simply reading data, build a JSON payload.
3. Transmit the JSON payload using an HTTPS call for the Universal Resource Locater (URL) for the resource.
4. Consume the returned JSON response.
5. If you make configuration changes, deploy the changes. See Deploying Configuration Changes, on page 45.

CHAPTER 4

Authenticating Your REST API Client Using OAuth

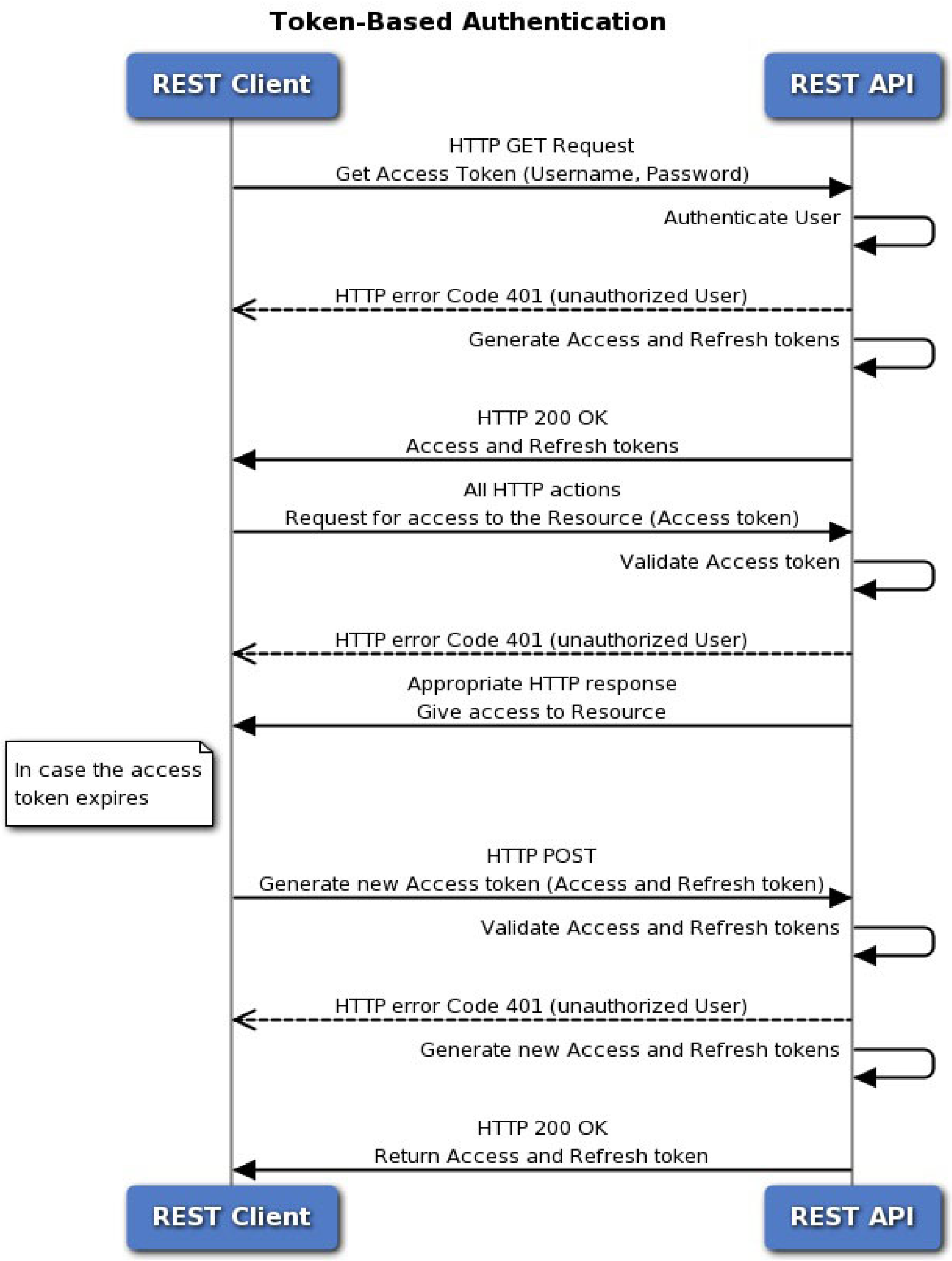
The threat defense REST API uses Oauth 2.0 for authenticating calls from API clients. OAuth is an access token-based method, and the threat defense uses JSON web tokens for the schema. The relevant standards are:

- RFC6749, The OAuth 2.0 Authorization Framework, https://tools.ietf.org/html/rfc6749.
- RFC7519, JSON Web Token (JWT), https://tools.ietf.org/html/rfc7519.

The following topics explain the methods for obtaining and using the required tokens.

Overview of the API Client Authentication Process

Following is the end-to-end view of how to authenticate your API client with the threat defense device.


Token-Based Authentication
REST Client
REST API
HTTP GET Request
Get Access Token (Username, Password)
Authenticate User
HTTP error Code 401 (unauthorized User)
Generate Access and Refresh tokens
HTTP 200 OK
Access and Refresh tokens
All HTTP actions
Request for access to the Resource (Access token)
Validate Access token
HTTP error Code 401 (unauthorized User)
Appropriate HTTP response
Give access to Resource
In case the access token expires
HTTP POST
Generate new Access token (Access and Refresh token)
Validate Access and Refresh tokens
HTTP error Code 401 (unauthorized User)
Generate new Access and Refresh tokens
HTTP 200 OK
Return Access and Refresh token
REST Client
REST API

Before you begin

Each token represents an HTTPS login session, which counts for API sessions and device manager sessions. There can be a maximum of 5 active HTTPS sessions. If you exceed this limit, the oldest session, either the device manager login or API token, is expired to allow the new session. Thus, it is important that you get only those tokens you need, and you reuse each token until it is expired, then renew them. Getting a new token for each API call will result in severe session churn and could lock users out of the device manager. These limits do not apply to SSH sessions.

Procedure

Step 1 Authenticate the API client user using whatever method you require.

Your client is obligated to authenticate users and ensure they have the authority to access and modify the threat defense device. If you want to provide differential abilities based on authorization rights, you need to build that into your client.

For example, if you want to allow read-only access, you must set up the required authentication server, user accounts, and so forth. Then, when a user with read-only rights logs into your client, you must ensure that you issue GET calls only. In API v1, this type of variable access cannot be controlled by the threat defense device itself. Starting with API v2, if you are using external users and you do not groom calls based on user authorization, you will get errors if there is a mismatch between user authorization and the calls you attempt.

For v1, when communicating with the device, you must use the **admin** user account on the threat defense device. The **admin** account has full read/write authorization for all user-configurable objects.

Step 2 Request a password-granted access token based on username/password using the **admin** account.

See Requesting a Password-Granted Access Token, on page 15.

Step 3 Optionally, request a custom access token for your client.

With a custom token, you can explicitly request a validity period, and assign a subject name for the token. See Requesting a Custom Access Token, on page 17.

Step 4 Use the access token on API calls in the Authorization: Bearer header.

See Using an Access Token on API Calls, on page 19.

Step 5 Before the access token expires, refresh the token.

See Refreshing an Access Token, on page 19.

Step 6 When you are finished, revoke the token if it has not yet expired.

See Revoking an Access Token, on page 21.

Requesting a Password-Granted Access Token

Every REST API call must include an authentication token to verify that the caller is authorized to perform the requested action. Initially, you need to obtain an access token by supplying the **admin** username/password. This is called a password-granted access token, that is, grant_type = password.

Procedure

Step 1 Create the JSON object for the password-granted access token grant.

```
{
  "grant_type": "password",
  "username": "string",
  "password": "string"
}
```

Specify the **admin** username and the correct password, for example:

```
{
  "grant_type": "password",
  "username": "admin",
  "password": "Admin123"
}
```

Step 2 Use POST /fdm/token to obtain the access token.

For example, the **curl** command would look like the following:

```
curl -X POST --header 'Content-Type: application/json' --header
'Accept: application/json' -d '{
   "grant_type": "password",
   "username": "admin",
   "password": "Admin123"
 }' 'https://ftd.example.com/api/fdm/latest/fdm/token'
```

Step 3 Retrieve the access and refresh tokens from the response.

A good response (status code 200) looks like the following:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE1MDI4MzI2NjcsInN
1YiI6ImFkbWluIiwianRpIjoiMGM3ZDBmNDgtODIwMS0xMWU3LWE4MWMtMDcwZmYzOW
U3ZjQ0IiwibmJmIjoxNTAyODMyNjY3LCJleHAiOjE1MDI4MzQ0NjcsInJlZnJlc2hU
b2tlbkV4cGlyZXNBdCI6MTUwMjgzNTA2NzQxOSwidG9rZW5UeXBlIjoiSldUX0FjY2
VzcyIsIm9yaWdpbiI6InBhc3N3b3JkIn0.b2hI6fVA_GbmhCOPM-ZUx6IC8SgCk1Ak
HXI-1lV0r7s",
  "expires_in": 1800,
  "token_type": "Bearer",
  "refresh_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE1MDI4MzI2NjcsI
nN1YiI6ImFkbWluIiwia nRpIjoiMGM3ZDBmNDgtODIwMS0xMWU3LWE4MWMtMDcwZmY
zOWU3ZjQ0IiwibmJmIjoxNTAyODMyNjY3LCJleHAiOjE1MDI4MzUwNjcsImFjY2Vzc1
Rva2VuRXhwaXJlc0F0IjoxNTAyODM0NDY3NDE5LCJyZWZyZXNoQ291bnQiOi0xLCJ0b2
tlblR5cGUiOiJKV1RfUmVmcmVzaCIsIm9yaWdpbiI6InBhc3N3b3JkIn0.iLNqz1c1Xl
vcq0j9pQYW4gwYsvUCcSyaiDRXGutAz_o",
  "refresh_expires_in": 2400
}
```

Where:

- **access_token** is the bearer token you need to include on API calls. See Using an Access Token on API Calls, on page 19.
- **expires_in** is the number of seconds for which the access token is valid, from the time the token is issued.
- **refresh_token** is the token you would use on a refresh request. See Refreshing an Access Token, on page 19.
- **refresh_expires**_in is the number of seconds for which the refresh token is valid. This is always longer than the access token validity period.

Requesting a Custom Access Token

You can use the password-granted access token. However, you can also request a custom access token. With a custom token, you can supply a subject name to help differentiate token usage (for your own purposes). You can also request specific validity periods if the default values returned for password tokens do not fit your requirements.

Before you begin

You must first get a password-granted access token before getting a custom token. See Requesting a Password-Granted Access Token, on page 15.

In addition:

- You can request a custom token only if you are a local user. External users cannot request custom tokens.
- You can use a custom token on the unit for which you obtain it only. You cannot use the token on the peer device in a high availability group.

Procedure

Step 1 Create the JSON object for the custom access token grant.

```
{
  "grant_type": "custom_token",
  "access_token": "string",
  "desired_expires_in": 0,
  "desired_refresh_expires_in": 0,
  "desired_subject": "string",
  "desired_refresh_count": 0
}
```

Where:

- **access_token** is a valid password-granted access token.
- **desired_expires_in** is an integer representing the number of seconds for which the custom access token will be valid. In comparison, the password-granted tokens are valid for 1800 seconds.
- **desired_refresh_expires_in** is an integer representing the number of seconds for which the custom refresh token will be valid. If you obtain a refresh token, ensure that this value is larger than the **desired_expires_in** value. In comparison, the password-granted refresh tokens are valid for 2400 seconds. This parameter is not required if you specify 0 for **desired_refresh_count**.
- **desired_subject** is a name you give to the custom token.
- **desired_refresh_count** is the number of times you want to be able to refresh the token. Specify 0 if you do not want to get a refresh token. When you do not have a refresh token, you must obtain a new access token when the existing one expires.

For example, the following requests a custom token for api-client that expires in 2400 seconds, with a refresh token that expires in 3000 seconds. The token can be refreshed 3 times.

```
{{
  "grant_type": "custom_token",
  "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE1MDI4MzI2NjcsInN
1YiI6ImFkbWluIiwianRpIjoiMGM3ZDBmNDgtODIwMS0xMWU3LWE4MWMtMDcwZmYzOW
U3ZjQ0IiwibmJmIjoxNTAyODMyNjY3LCJleHAiOjE1MDI4MzQ0NjcsInJlZnJlc2hU
b2tlbkV4cGlyZXNBdCI6MTUwMjgzNTA2NzQxOSwidG9rZW5UeXBlIjoiSldUX0FjY2
VzcyIsIm9yaWdpbiI6InBhc3N3b3JkIn0.b2hI6fVA_GbmhCOPM-ZUx6IC8SgCk1Ak
HXI-llV0r7s",
  "desired_expires_in": 2400,
  "desired_refresh_expires_in": 3000,
  "desired_subject": "api-client",
  "desired_refresh_count": 3
}
```

Step 2 Use POST /fdm/token to obtain the access token.

For example, the **curl** command would look like the following:

```
curl -X POST --header 'Content-Type: application/json' --header
'Accept: application/json' -d '{
   "grant_type": "custom_token",
   "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE1MDI4MzU5N
jgsInN1YiI6ImFkbWluIiwianRpIjoiYmMyNjM4N2EtODIwOC0xMWU3LWE4MWM
tYzNlYTZkZjJjZThjIiwibmJmIjoxNTAyODM1OTY4LCJleHAiOjE1MDI4Mzc3N
jgsInJlZnJlc2hUb2tlbkV4cGlyZXNBdCI6MTUwMjgzODM2ODYwNiwidG9rZW5
UeXBlIjoiSldUX0FjY2VzcyIsIm9yaWdpbiI6InBhc3N3b3JkIn0.acOE_Y4SE
ds-NE4Qw99fQlUzdoSkhsjInaCh0a9WK38",
   "desired_expires_in": 2400,
   "desired_refresh_expires_in": 3000,
   "desired_subject": "api-client",
   "desired_refresh_count": 3
 }' 'https://ftd.example.com/api/fdm/latest/fdm/token'
```

Step 3 Retrieve the access and refresh tokens from the response.

A good response (status code 200) looks like the following:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE1MDI4MzU5O
TEsInN1YiI6ImFwaS1jbGllbnQiLCJqdGkiOiJjOWIxYzdjYi04MjA4LTExZT
ctYTgxYy02YmY0NzY3ZmRmZGUiLCJuYmYiOjE1MDI4MzU5OTEsImV4cCI6MTU
wMjgzODM5MSwicmVmcmVzaFRva2VuRXhwaXJlc0F0IjoxNTAyODM4OTkxMzMx
LCJ0b2tlblR5cGUiOiJKV1RfQWNjZXNzIiwib3JpZ2luIjoiY3VzdG9tIn0.9
IVzLjGffVQffHAWdrNkrYfvuO6TgpJ7Zi_z3RYubN8",
  "expires_in": 2400,
  "token_type": "Bearer",
  "refresh_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE1MDI4MzU5
OTEsInN1YiI6ImFwaS1jbGllbnQiLCJqdGkiOiJjOWIxYzdjYi04MjA4LTExZ
TctYTgxYy02YmY0NzY3ZmRmZGUiLCJuYmYiOjE1MDI4MzU5OTEsImV4cCI6MT
UwMjgzODk5MSwiYWNjZXNzVG9rZW5FeHBpcmVzQXQiOjE1MDI4MzgzOTEzMzE
sInJlZnJlc2hDb3VudCI6MywidG9rZW5UeXBlIjoiSldUX1JlZnJlc2giLCJv
cmlnaW4iOiJjdXN0b20ifQ.qseqjg3Uo183YvfN_77iJZELEqwpWw5AbKAqAn
CIcSA",
  "refresh_expires_in": 3000
}
```

Where:

- **access_token** is the bearer token you need to include on API calls. See Using an Access Token on API Calls, on page 19.
- **expires_in** is the number of seconds for which the access token is valid, from the time the token is issued.
- **refresh_token** is the token you would use on a refresh request. See Refreshing an Access Token, on page 19.
- **refresh_expires**_in is the number of seconds for which the refresh token is valid. This is always longer than the access token validity period.

Using an Access Token on API Calls

After you obtain either a password-granted or custom access token, you must include it on each API call in the **Authorization: Bearer** header to the HTTPS request.

For example, a **curl** command to perform GET /object/networks might look like the following:

```
curl -k -X GET -H 'Accept: application/json'
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.yJp
YXQiOjE1MDI4MzU5OTEsInN1YiI6ImFwaS1jbGllbnQiLCJqdG
kiOiJjOWIxYzdjYi04MjA4LTExZTctYTgxYy02YmY0NzY3ZmRm
ZGUiLCJuYmYiOjE1MDI4MzU5OTEsImV4cCI6MTUwMjgzODM5MS
wicmVmcmVzaFRva2VuRXhwaXJlc0F0IjoxNTAyODM4OTkxMzMx
LCJ0b2tlblR5cGUiOiJKV1RfQWNjZXNzIiwib3JpZ2luIjoiY3
VzdG9tIn0.9IVzLjGffVQffHAWdrNkrYfvuO6TgpJ7Zi_z3RYu
bN8'
'https://ftd.example.com/api/fdm/latest/object/networks'
```

Note When you use the API Explorer to try out methods and resources, the **curl** command shown does not include the **Authorization: Bearer** header. However, you must add this header when making calls from your API client.

Refreshing an Access Token

After an access token expires, you need to refresh it using the refresh token that was supplied in the original grant. A refreshed access token is actually different than the original access token. “Refreshing” actually supplies a new pair of access token and refresh token, it does not simply extend the life of the old access token.

Procedure

Step 1 Create the JSON object for the refresh token grant.

```
{
  "grant_type": "refresh_token",
  "refresh_token": "string"
}
```

The **refresh_token** can be from a password-granted or custom access token grant.

For example:

```
{
  "grant_type": "refresh_token",
  "refresh_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQ
iOjE1MDI4MzU5OTEsInN1YiI6ImFwaS1jbGllbnQiLCJqdGk
iOiJjOWIxYzdjYi04MjA4LTExZTctYTgxYy02YmY0NzY3ZmR
mZGUiLCJuYmYiOjE1MDI4MzU5OTEsImV4cCI6MTUwMjgzODk
5MSwiYWNjZXNzVG9rZW5FeHBpcmVzQXQiOjE1MDI4MzgzOTE
zMzEsInJlZnJlc2hDb3VudCI6MywidG9rZW5UeXBlIjoiSld
UX1JlZnJlc2giLCJvcmlnaW4iOiJjdXN0b20ifQ.qseqjg3U
o183YvfN_77iJZELEqwpWw5AbKAqAnCIcSA"
}
```

Step 2 Use POST /fdm/token to obtain the refreshed access token.

For example, the **curl** command would look like the following:

```
curl -X POST --header 'Content-Type: application/json' --header
'Accept: application/json' -d '{
   "grant_type": "refresh_token",
   "refresh_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE1M
DI4MzU5OTEsInN1YiI6ImFwaS1jbGllbnQiLCJqdGkiOiJjOWIxYzdj
Yi04MjA4LTExZTctYTgxYy02YmY0NzY3ZmRmZGUiLCJuYmYiOjE1MDI
4MzU5OTEsImV4cCI6MTUwMjgzODk5MSwiYWNjZXNzVG9rZW5FeHBpcm
VzQXQiOjE1MDI4MzgzOTEzMzEsInJlZnJlc2hDb3VudCI6MywidG9rZ
W5UeXBlIjoiSldUX1JlZnJlc2giLCJvcmlnaW4iOiJjdXN0b20ifQ.q
seqjg3Uo183YvfN_77iJZELEqwpWw5AbKAqAnCIcSA"
 }' 'https://ftd.example.com/api/fdm/latest/fdm/token'
```

Step 3 Retrieve the access and refresh tokens from the response.

A good response (status code 200) looks like the following. In this example, the refresh token was for a custom token. The expiration periods are based on the values from the original custom access token request.

```
{
  "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQ
iOjE1MDI4Mzc1MTAsInN1YiI6ImFwaS1jbGllbnQiLCJqdG
kiOiJjOWIxYzdjYi04MjA4LTExZTctYTgxYy02YmY0NzY3Z
mRmZGUiLCJuYmYiOjE1MDI4Mzc1MTAsImV4cCI6MTUwMjgz
OTkxMSwicmVmcmVzaFRva2VuRXhwaXJlc0F0IjoxNTAyODQ
wNTEwNzQxLCJ0b2tlblR5cGUiOiJKV1RfQWNjZXNzIiwib3
JpZ2luIjoiY3VzdG9tIn0.fAAreX0DdnuqnM0Bs0NXYnI-9
jkpyW1pWDMwgwO_h7A",
  "expires_in": 2400,
  "token_type": "Bearer",
  "refresh_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYX
QiOjE1MDI4Mzc1MTAsInN1YiI6ImFwaS1jbGllbnQiLCJqd
GkiOiJjOWIxYzdjYi04MjA4LTExZTctYTgxYy02YmY0NzY3
ZmRmZGUiLCJuYmYiOjE1MDI4Mzc1MTAsImV4cCI6MTUwMjg
0MDUxMCwiYWNjZXNzVG9rZW5FeHBpcmVzQXQiOjE1MDI4Mz
k5MTEwNzIsInJlZnJlc2hDb3VudCI6MiwidG9rZW5UeXBlI
joiSldUX1JlZnJlc2giLCJvcmlnaW4iOiJjdXN0b20ifQ.p
Adc2N0oun7Yyw872qK12pFlix4arAwyMETD1ErKu5c",
  "refresh_expires_in": 3000
}
```

Where:

- **access_token** is the bearer token you need to include on API calls. See Using an Access Token on API Calls, on page 19.
- **expires_in** is the number of seconds for which the access token is valid, from the time the token is issued.
- **refresh_token** is the token you would use on a refresh request.
- **refresh_expires**_in is the number of seconds for which the refresh token is valid. This is always longer than the access token validity period.

Revoking an Access Token

Because access tokens are valid for a particular length of time, you should clean up by revoking a token when the user logs out of your API client. This ensures that no back door is left open into the threat defense device.

Procedure

Step 1 Create the JSON object for the revoke token grant.

```
{
  "grant_type": "revoke_token",
  "access_token": "string",
  "token_to_revoke": "string",
  "custom_token_id_to_revoke": "string",
  "custom_token_subject_to_revoke": "string"
}
```

Where:

- **access_token** must be a password-granted access token. You cannot revoke a token using a custom access token.
- You must specify one, and only one, of the following:
  - **token_to_revoke** is a password-granted or custom token that you want to revoke. This can be the same token as **access_token**, so you can use a password-granted token to revoke itself.
  - (Do not use.) **custom_token_id_to_revoke** identifies custom access token by its internal unique ID. However, there is no direct way for you to obtain this value. Use the other options instead.
  - **custom_token_subject_to_revoke** is the **desired_subject** value for the custom access token that you want to revoke.

For example:

```
{
  "grant_type": "revoke_token",
  "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQ
iOjE1MDI5MDQzMjQsInN1YiI6ImFkbWluIiwianRpIjoiZT
MzNGIxOWYtODJhNy0xMWU3LWE4MWMtNGQ3NzY2ZTExMzVkI
iwibmJmIjoxNTAyOTA0MzI0LCJleHAiOjE1MDI5MDYxMjQs
InJlZnJlc2hUb2tlbkV4cGlyZXNBdCI6MTUwMjkwNjcyNDE
xMiwidG9rZW5UeXBlIjoiSldUX0FjY2VzcyIsIm9yaWdpbi
I6InBhc3N3b3JkIn0.OVZBT9yVZc4zxZfZiiLH4SZcFclaH
yCPbZJC_Gyd5FE",
  "custom_token_subject_to_revoke": "api-client"
}
```

Step 2 Use POST /fdm/token to revoke the access token.

For example, the **curl** command would look like the following:

```
curl -X POST --header 'Content-Type: application/json'
--header 'Accept: application/json' -d '{
   "grant_type": "revoke_token",
   "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQ
iOjE1MDI5MDQzMjQsInN1YiI6ImFkbWluIiwianRpIjoiZTM
zNGIxOWYtODJhNy0xMWU3LWE4MWMtNGQ3NzY2ZTExMzVkIiw
ibmJmIjoxNTAyOTA0MzI0LCJleHAiOjE1MDI5MDYxMjQsInJ
lZnJlc2hUb2tlbkV4cGlyZXNBdCI6MTUwMjkwNjcyNDExMiw
idG9rZW5UeXBlIjoiSldUX0FjY2VzcyIsIm9yaWdpbiI6InB
hc3N3b3JkIn0.OVZBT9yVZc4zxZfZiiLH4SZcFclaHyCPbZJ
C_Gyd5FE",
   "custom_token_subject_to_revoke": "api-client"
 }' 'https://ftd.example.com/api/fdm/latest/fdm/token'
```

Step 3 Evaluate the response to verify that the token was revoked.

A good response (status code 200) looks like the following.

```
{
  "message": "OK",
  "status_code": 200
}
```

CHAPTER 5

Configuring External Users for the API

Version Requirement: To use external AAA, you must be running the threat defense version 6.3(0) or higher, and the threat defense REST API v2 or higher.

You can configure the device to use an external RADIUS AAA server to authenticate and authorize user access to the threat defense REST API. You can use RADIUS user accounts instead of, or in addition to, the built-in local **admin** user account.

When using external AAA, you can define accounts to have different authorization levels. This lets you limit who can make configuration changes to the device while still providing read-only access to support staff.

The following procedure explains the end-to-end process of setting up the RADIUS accounts and then configuring the device to use external AAA for authentication and authorization.

Before you begin

Keep the following operational factors in mind when using external authorization.

- If the device is configured for high availability, configure external authorization on the active unit. You must then run a deployment job for the authorization settings to allow user access to the standby device.
- Each time a new user accesses the system, a User resource is created for that user. You need to deploy the configuration to save that user object.

  (Versions prior to the threat defense 6.6.) If you are operating in high availability (HA) mode, you must deploy the configuration before that user can log into the standby unit. Because only admin or read-write users can start a deployment job, a first-time read-only user must have someone else deploy the configuration to save the User object.

  Starting with the threat defense 6.6, the HA restrictions are removed. An external user can log into the standby unit without first logging into the active unit and deploying the configuration. The user object will not be created on the standby unit, but user characteristics will be cached and the user will be given access, assuming a valid username/password is provided.

Procedure

Step 1 Define Authorization Rights in the RADIUS User Accounts, on page 24.

Step 2 Define the RADIUS Servers, on page 24.

Step 3 Create a AAA Server Group for the RADIUS Servers, on page 26.

Step 4 Establish the AAA Server Group as an Authentication Source for HTTPS Access, on page 28.

Step 5 Use **POST /operational/deploy** to start a deployment job.

The **curl** command would be similar to the following:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json'
'https://ftd.example.com/api/fdm/latest/operational/deploy'
```

For detailed information on deploying changes, see Deploying Configuration Changes, on page 45.

Step 6 Verify External User Access, on page 31.

- Define Authorization Rights in the RADIUS User Accounts, on page 24
- Define the RADIUS Servers, on page 24
- Create a AAA Server Group for the RADIUS Servers, on page 26
- Establish the AAA Server Group as an Authentication Source for HTTPS Access, on page 28
- Verify External User Access, on page 31

Define Authorization Rights in the RADIUS User Accounts

You can provide access to the threat defense REST API from an external RADIUS server. By enabling RADIUS authentication and authorization, you can provide different levels of access rights, and not have every user log in through the local **admin** account.

Note These external users are also authorized for device manager.

To provide role-based access control (RBAC), update the user accounts on your RADIUS server to define the **cisco-av-pair** attribute. This attribute must be defined correctly on a user account, or the user is denied access to the REST API. Following are the supported values for the cisco-av-pair attribute:

- **fdm.userrole.authority.admin** provides full Administrator access. These users can do all actions that the local **admin** user can do.
- **fdm.userrole.authority.rw** provides read-write access. These users can do everything a read-only user can do, and also edit and deploy the configuration. The only restrictions are for system-critical actions, which include installing upgrades, creating and restoring backups, viewing the audit log, and logging off other users.
- **fdm.userrole.authority.ro** provides read-only access. These users can view dashboards and the configuration, but cannot make any changes. If the user tries to make a change, the error message explains that this is due to lack of permission.

Define the RADIUS Servers

After you configure the user accounts in the RADIUS server to define the appropriate authorization rights, you can configure the device to use the server to authenticate and authorize access to the REST API.

Use the **POST /object/radiusidentitysources** resource to create an object for each RADIUS server you want to define.

Procedure

Step 1 Create the JSON object body for the RADIUS server.

Following is an example of the JSON object to use with this call.

```
{
  "name": "aaa-server-1",
  "description": "RADIUS server for API access.",
  "host": "172.16.246.220",
  "timeout": 10,
  "serverAuthenticationPort": 1812,
  "serverSecretKey": "secret123",
  "type": "radiusidentitysource"
}
```

The attributes are:

- **name**—The object name. This does not need to match anything defined on the RADIUS server.
- **description**—(Optional.) A description of the object.
- **host**—The IP address or fully-qualified host name of the RADIUS server.
- **timeout**—(Optional.) The length of time, 1-300 seconds, that the system waits for a response from the server before sending the request to the next server. If you do not include this attribute, the default is 10 seconds.
- **serverAuthenticationPort**—(Optional.) The port on which RADIUS authentication and authorization are performed. If you do not include this attribute, the default is 1812.
- **serverSecretKey**—(Optional.) The shared secret that is used to encrypt data between the threat defense device and the RADIUS server. The key is a case-sensitive, alphanumeric string of up to 64 characters, with no spaces. The key must start with an alphanumeric character or an underscore, and it can contain the special characters: $ & - _ . + @. The string must match the one configured on the RADIUS server. If you do not configure a secret key, the connection is not encrypted.

Step 2 Post the object.

For example, the **curl** command would look like the following:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json'
 -d '{
    "name": "aaa-server-1",
   "description": "RADIUS server for API access.",
   "host": "172.16.246.220",
   "timeout": 10,
   "serverAuthenticationPort": 1812,
   "serverSecretKey": "secret123",
   "type": "radiusidentitysource"
 }' 'https://ftd.example.com/api/fdm/latest/object/radiusidentitysources'
```

Step 3 Verify the response.

You should get a response code of 200. A successful response body would look something like the following. Note that sensitive information, such as the secret key, is masked in the response.

```
{
  "version": "nfamb3cr2jlyi",
  "name": "aaa-server-1",
  "description": "RADIUS server for API access.",
  "host": "172.16.246.220",
  "timeout": 10,
  "serverAuthenticationPort": 1812,
  "serverSecretKey": "*********",
  "capabilities": [
    "AUTHENTICATION",
    "AUTHORIZATION"
  ],
  "id": "1b962e3b-6e56-11e8-bd65-379fa8aaaba1",
  "type": "radiusidentitysource",
  "links": {
    "self": "https://ftd.example.com/api/fdm/latest/object/
radiusidentitysources/1b962e3b-6e56-11e8-bd65-379fa8aaaba1"
  }
}
```

Create a AAA Server Group for the RADIUS Servers

After you create the RADIUS server objects, use the **POST /object/radiusidentitysourcegroups** resource to create a AAA group to contain the radiusidentitysource objects.

You can add up to 16 RADIUS servers to a RADIUS AAA server group. These servers must be backups of each other: that is, they need to have the same list of user accounts.

Procedure

Step 1 Create the JSON object body for the RADIUS server group.

Following is an example of the JSON object to use with this call.

```
{
  "name": "radius-group",
  "maxFailedAttempts": 3,
  "deadTime": 10,
  "description": "AAA RADIUS server group.",
  "radiusIdentitySources": [
    {
      "id": "1b962e3b-6e56-11e8-bd65-379fa8aaaba1",
      "type": "radiusidentitysource",
      "version": "nfamb3cr2jlyi",
      "name": "aaa-server-1"
    }
  ],
  "type": "radiusidentitysourcegroup"
}
```

The attributes are:

- **name**—The object name. This does not need to match anything defined on the member RADIUS servers.

- **maxFailedAttempts**—(Optional.) Failed servers are reactivated only after all servers have failed. The dead time is how long to wait, from 0 - 1440 minutes, after the last server fails before reactivating all servers. If you do not include this attribute, the default is 10 minutes.
- **deadTime**—(Optional.) The number of failed requests (that is, requests that do not get a response) sent to a RADIUS server in the group before trying the next server. You can specify 1-5, and the default is 3. When the maximum number of failed attempts is exceeded, the system marks the server as Failed.

  For a given feature, if you configured a fallback method using the local database, and all the servers in the group fail to respond, then the group is considered to be unresponsive, and the fallback method is tried. The server group remains marked as unresponsive for the duration of the dead time, so that additional AAA requests within that period do not attempt to contact the server group, and the fallback method is used immediately.
- **description**—(Optional.) A description of the object.
- **radiusIdentitySources**—This is a group of items that define each radiusidentitysource object that defines a RADIUS server to include in the group. Include the items within the [brackets]. Following are the attributes and syntax for each object. You get the value for the **id**, **version**, and **name** attributes from the individual objects; the information is in the response body when you create the objects. You can also get the information from a **GET /object/radiusidentitysources** call. The **type** must be **radiusidentitysource**.

```
{
  "id": "1b962e3b-6e56-11e8-bd65-379fa8aaaba1",
  "type": "radiusidentitysource",
  "version": "nfamb3cr2jlyi",
  "name": "aaa-server-1"
}
```

Step 2 Post the object.

For example, the **curl** command would look like the following:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json'
 -d '{
   "name": "radius-group",
   "maxFailedAttempts": 3,
   "deadTime": 10,
   "description": "AAA RADIUS server group.",
   "radiusIdentitySources": [
     {
       "id": "1b962e3b-6e56-11e8-bd65-379fa8aaaba1",
       "type": "radiusidentitysource",
       "version": "nfamb3cr2jlyi",
       "name": "aaa-server-1"
     }
   ],
   "type": "radiusidentitysourcegroup"
 }' 'https://ftd.example.com/api/fdm/latest/object/radiusidentitysourcegroups'
```

Step 3 Verify the response.

You should get a response code of 200. A successful response body would look something like the following.

```
{
  "version": "7r572novdiyy",
  "name": "radius-group",
  "maxFailedAttempts": 3,
  "deadTime": 10,
  "description": "AAA RADIUS server group.",
  "radiusIdentitySources": [
    {
      "version": "nfamb3cr2jlyi",
      "name": "aaa-server-1",
      "id": "1b962e3b-6e56-11e8-bd65-379fa8aaaba1",
      "type": "radiusidentitysource"
    }
  ],
  "activeDirectoryRealm": null,
  "id": "0a7996ae-6e5b-11e8-bd65-dbab801c44b9",
  "type": "radiusidentitysourcegroup",
  "links": {
    "self": "https://ftd.example.com/api/fdm/latest/object/
radiusidentitysourcegroups/0a7996ae-6e5b-11e8-bd65-dbab801c44b9"
  }
}
```

Establish the AAA Server Group as an Authentication Source for HTTPS Access

Use the **PUT /devicesettings/default/aaasettings/{objId}** resource to identify the RADIUS AAA server group as an identity source for user authorization.

There is no POST method: the objects needed for system authentication already exist. You must first do a GET to determine the relevant ID and version values.

Procedure

Step 1 Use **GET /devicesettings/default/aaasettings** to determine the attributes of the aaasettings objects.

The **curl** command would be similar to the following:

```
curl -X GET --header 'Accept: application/json'
'https://ftd.example.com/api/fdm/latest/devicesettings/default/aaasettings'
```

For example, the response body should be similar to the following. This example shows that the local identity source is the one defined for HTTPS access. It is also used for SSH access, which is not relevant for the REST API.

```
{
  "items": [
    {
      "version": "du52clrtmawlt",
      "name": "HTTPS",
      "identitySourceGroup": {
        "version": "cynutari5ffkl",
        "name": "LocalIdentitySource",
        "id": "e3e74c32-3c03-11e8-983b-95c21a1b6da9",
        "type": "localidentitysource"
      },
      "description": null,
      "protocolType": "HTTPS",
      "useLocal": "NOT_APPLICABLE",
      "id": "00000003-0000-0000-0000-000000000007",
      "type": "aaasetting",
      "links": {
        "self": "https://ftd.example.com/api/fdm/latest/
devicesettings/default/aaasettings/00000003-0000-0000-0000-000000000007"
      }
    },
    {
      "version": "fgkhvu4kwucgv",
      "name": "SSH",
      "identitySourceGroup": {
        "version": "cynutari5ffkl",
        "name": "LocalIdentitySource",
        "id": "e3e74c32-3c03-11e8-983b-95c21a1b6da9",
        "type": "localidentitysource"
      },
      "description": null,
      "protocolType": "SSH",
      "useLocal": "NOT_APPLICABLE",
      "id": "00000003-0000-0000-0000-000000000008",
      "type": "aaasetting",
      "links": {
        "self": "https://ftd.example.com/api/fdm/latest/
devicesettings/default/aaasettings/00000003-0000-0000-0000-000000000008"
      }
    }
  ],
  "paging": {
    "prev": [],
    "next": [],
    "limit": 10,
    "offset": 0,
    "count": 2,
    "pages": 0
  }
}
```

Step 2 (Optional.) Use **GET /devicesettings/default/aaasettings/{objId}** to obtain a copy of the HTTPS AAA setting object to narrow your view.

Your PUT call will update the HTTPS object only. You do not need to update the SSH object.

In this example, the ID of the HTTPS object is 00000003-0000-0000-0000-000000000007, so a **curl** command would be similar to the following:

```
curl -X GET --header 'Accept: application/json'
'https://ftd.example.com/api/fdm/latest/devicesettings/
default/aaasettings/00000003-0000-0000-0000-000000000007'
```

The response body would be similar to the following.

```
{
  "version": "ha4653ootep7z",
  "name": "HTTPS",
  "identitySourceGroup": {
    "version": "cynutari5ffkl",
    "name": "LocalIdentitySource",
    "id": "e3e74c32-3c03-11e8-983b-95c21a1b6da9",
    "type": "localidentitysource"
  },
  "description": null,
  "protocolType": "HTTPS",
  "useLocal": "NOT_APPLICABLE",
  "id": "00000003-0000-0000-0000-000000000007",
  "type": "aaasetting",
  "links": {
    "self": "https://ftd.example.com/api/fdm/latest/
devicesettings/default/aaasettings/00000003-0000-0000-0000-000000000007"
  }
```

Step 3 Create the JSON object body for AAA management access.

Following is an example of the JSON object to use with this call.

```
{
  "version": "ha4653ootep7z",
  "name": "HTTPS",
  "identitySourceGroup": {
    "id": "0a7996ae-6e5b-11e8-bd65-dbab801c44b9",
    "type": "radiusidentitysourcegroup",
    "version": "7r572novdiyy",
    "name": "radius-group"
  },
  "description": null,
  "protocolType": "HTTPS",
  "useLocal": "BEFORE",
  "id": "00000003-0000-0000-0000-000000000007",
  "type": "aaasetting"
}
```

The attributes are:

- **version**—The version of the HTTPS object. Find this value in the response body for the GET call.
- **name**—The object name, **HTTPS**. Find this value in the response body for the GET call.
- **identitySourceGroup**—This identifies the RADIUS server group. Obtain the **id**, **version**, and **name** values from the response body when you created the server group (or a **GET /object/radiusidentitysourcegroups** call). The type must be **radiusidentitysourcegroup**.
- **description**—(Optional.) A description of the object.
- **protocolType**—The protocol to which this source applies, **HTTPS**.
- **useLocal**—How to use the local identity source, which contains the local admin user account. Enter one of the following options:
  - **Before**—The system checks the username and password against the local source first.
  - **After**—The local source is checked only if the external source is unavailable or if the user account was not found in the external source.
  - **Never**—(Not recommended.) The local source is never used, so you cannot log in as the **admin** user.

    Caution If you select **Never**, you will not be able to log into the device manager or use the API using the **admin** account. You will be locked out of the system if the RADIUS server becomes unavailable, or if you miss-configure the accounts in the RADIUS server.

- **id**—The ID value for the HTTPS object. Find this value in the response body for the GET call.
- **type**—The object type, **aaasetting**.

Step 4 Put the object.

For example, the **curl** command would look like the following. Note that the {objId} in the URL and the id for the aaasettings object in the JSON object are the same.

```
curl -X PUT --header 'Content-Type: application/json' --header 'Accept: application/json'
-d '{
   "version": "ha4653ootep7z",
   "name": "HTTPS",
   "identitySourceGroup": {
     "id": "0a7996ae-6e5b-11e8-bd65-dbab801c44b9",
     "type": "radiusidentitysourcegroup",
     "version": "7r572novdiyy",
     "name": "radius-group"
   },
   "description": null,
   "protocolType": "HTTPS",
   "useLocal": "BEFORE",
   "id": "00000003-0000-0000-0000-000000000007",
   "type": "aaasetting"
 }' 'https://ftd.example.com/api/fdm/latest/devicesettings/
default/aaasettings/00000003-0000-0000-0000-000000000007'
```

Step 5 Verify the response.

You should get a response code of 200. A successful response body would look something like the following.

```
{
  "version": "ehxcytq4iccb3",
  "name": "HTTPS",
  "identitySourceGroup": {
    "version": "7r572novdiyy",
    "name": "radius-group",
    "id": "0a7996ae-6e5b-11e8-bd65-dbab801c44b9",
    "type": "radiusidentitysourcegroup"
  },
  "description": null,
  "protocolType": "HTTPS",
  "useLocal": "BEFORE",
  "id": "00000003-0000-0000-0000-000000000007",
  "type": "aaasetting",
  "links": {
    "self": "https://ftd.example.com/api/fdm/latest/devicesettings/
default/aaasettings/00000003-0000-0000-0000-000000000007"
  }
}
```

Verify External User Access

After the deployment job completes, you can test external user access to both the device manager and the REST API.

Procedure

Step 1 Log into the device manager using an external username that has a valid cisco-av-pair attribute.

The login should be successful, and the upper right of the page should show the username and privilege level.

Step 2 Obtain a REST API token for an external user.

If the user can obtain a token, the user can use the resources and methods allowed for the assigned privilege level.

a) Create the JSON object body for a simple password-granted token.

```
{
  "grant_type": "password",
  "username": "radiusreadwriteuser1",
  "password": "Readwrite123!"
}
```

b) Use **POST /fdm/token** to obtain the token.

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json'
 -d '{
   "grant_type": "password",
   "username": "radiusreadwriteuser1",
   "password": "Readwrite123!"
 }' 'https://ftd.example.com/api/fdm/latest/fdm/token'
```

c) Evaluate the response to verify that a token was granted.

You should get a response code of 200. Getting a token means that the system was able to authenticate the user.

```
{
  "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE1Mjg4MjM
3MTAsInN1YiI6InJhZGl1c3JlYWR3cml0ZXVzZXIxIiwianRpIjoiMjk5Zj
Q5YjYtNmU2NC0xMWU4LWJkNjUtNmY0ZmVmYjY1MzI5IiwibmJmIjoxNTI4O
DIzNzEwLCJleHAiOjE1Mjg4MjU1MTAsInJlZnJlc2hUb2tlbkV4cGlyZXNB
dCI6MTUyODgyNjExMDg4OSwidG9rZW5UeXBlIjoiSldUX0FjY2VzcyIsInV
zZXJVdWlkIjoiMjliMjBlNjctNmU2NC0xMWU4LWJkNjUtMzU4MmUwZjU5Yj
Q4IiwidXNlclJvbGUiOiJST0xFX1JFQURfV1JJVEUiLCJvcmlnaW4iOiJwY
XNzd29yZCJ9.dtKsl9IB4ds3RAktEeaSuQy_Zs2SrzLr976UtblBt28",
  "expires_in": 1800,
  "token_type": "Bearer",
  "refresh_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE1Mjg4Mj
M3MTAsInN1YiI6InJhZGl1c3JlYWR3cml0ZXVzZXIxIiwianRpIjoiMjk5Z
jQ5YjYtNmU2NC0xMWU4LWJkNjUtNmY0ZmVmYjY1MzI5IiwibmJmIjoxNTI4
ODIzNzEwLCJleHAiOjE1Mjg4MjYxMTAsImFjY2Vzc1Rva2VuRXhwaXJlc0F
0IjoxNTI4ODI1NTEwODg5LCJyZWZyZXNoQ291bnQiOi0xLCJ0b2tlblR5cG
UiOiJKV1RfUmVmcmVzaCIsInVzZXJVdWlkIjoiMjliMjBlNjctNmU2NC0xM
WU4LWJkNjUtMzU4MmUwZjU5YjQ4IiwidXNlclJvbGUiOiJST0xFX1JFQURf
V1JJVEUiLCJvcmlnaW4iOiJwYXNzd29yZCJ9.Lc7MYmieNMMrjx7XoTiW-x
8Z8qFCnzfNM1apgbwLQvo",
  "refresh_expires_in": 2400
}
```

Step 3 Use **GET /object/users** to verify that user objects were created for each user.

User objects are automatically created for each new user who logs into the device manager or who obtains an access token. You must run a deployment job to save these user objects. In high availability mode, the deployment job is required before the user can log into the standby unit.

For example, the **curl** command would look like the following:

```
curl -X GET --header 'Accept: application/json'
'https://ftd.example.com/api/fdm/latest/object/users'
```

The following response body shows that two external users logged in. Note that **userRole** shows the rights obtained from the cisco-av-pair configured in the RADIUS server for those user accounts. Use this information to verify you configured the RADIUS user accounts correctly. The **admin** user is the locally-defined user.

```
{
  "items": [
    {
      "version": "h2vom4wckm2js",
      "name": "radiusadminuser1",
      "password": null,
      "newPassword": null,
      "userPreferences": {
        "preferredTimeZone": "(UTC+00:00) UTC",
        "colorTheme": "NORMAL_CISCO_IDENTITY",
        "type": "userpreferences"
      },
      "userRole": "ROLE_ADMIN",
      "identitySourceId": "0a7996ae-6e5b-11e8-bd65-dbab801c44b9",
      "userServiceTypes": [
        "MGMT"
      ],
      "id": "150d9754-6e63-11e8-bd65-ed9b20f62114",
      "type": "user",
      "links": {
        "self": "https://ftd.example.com/api/fdm/latest/
object/users/150d9754-6e63-11e8-bd65-ed9b20f62114"
      }
    },
    {
      "version": "p4rgwcjr5colj",
      "name": "admin",
      "password": null,
      "newPassword": null,
      "userPreferences": {
        "preferredTimeZone": "(UTC-07:00) America/Los_Angeles",
        "colorTheme": "NORMAL_CISCO_IDENTITY",
        "type": "userpreferences"
      },
      "userRole": "ROLE_ADMIN",
      "identitySourceId": "e3e74c32-3c03-11e8-983b-95c21a1b6da9",
      "userServiceTypes": [
        "MGMT"
      ],
      "id": "5023d3ab-6dc5-11e8-b9ed-db6dba9bf94c",
      "type": "user",
      "links": {
        "self": "https://ftd.example.com/api/fdm/latest/
object/users/5023d3ab-6dc5-11e8-b9ed-db6dba9bf94c"
      }
    },
    {
      "version": "ngx7a2dixngoq",
      "name": "radiusreadwriteuser1",
      "password": null,
      "newPassword": null,
      "userPreferences": {
        "preferredTimeZone": "(UTC+00:00) UTC",
        "colorTheme": "NORMAL_CISCO_IDENTITY",
        "type": "userpreferences"
      },
      "userRole": "ROLE_READ_WRITE",
      "identitySourceId": "0a7996ae-6e5b-11e8-bd65-dbab801c44b9",
      "userServiceTypes": [
        "MGMT"
      ],
      "id": "29b20e67-6e64-11e8-bd65-3582e0f59b48",
      "type": "user",
      "links": {
        "self": "https://ftd.example.com/api/fdm/latest/
object/users/29b20e67-6e64-11e8-bd65-3582e0f59b48"
      }
    }
  ],
  "paging": {
    "prev": [],
    "next": [],
    "limit": 10,
    "offset": 0,
    "count": 3,
    "pages": 0
  }
}
```

CHAPTER 6

Using the Methods and Resources

The following topics explain in general how to use the various methods and resources.

Trying a Method and Interpreting the Results

You can use the API Explorer to test the various methods. This topic explains the general process, with an explanation of the response the system returns. See the topics on each method type for specific method-related techniques.

The **Try It Out!** button for each method/resource interacts directly with the system. GET retrieves actual data, POST/PUT creates or modifies real resources, and DELETE removes real objects. You are making real configuration changes in the system, although the changes are not immediately deployed. To make the changes active, use the POST /operational/deploy resource to start a deployment job.

You can find the **Try It Out!** button after the **Response Message** section when you open a method/resource. For some method/resources, you must enter an object ID to test it. In this case, you typically need to first do a GET on the parent resource. For more information, see Finding the Object ID (objId) and Parent ID, on page 8.

For POST/PUT, you also need to fill in the required values in the JSON model.

After you click **Try It Out!**, the API Explorer adds the results to the page after the button. The response includes the following sections:

Curl

The **curl** command used to make the call. For example, clicking **Try It Out!** on the GET /object/networks resource returns something like the following. The "v" element in the path changes with each new version of the API.

```
curl -X GET --header 'Accept: application/json'
'https://ftd.example.com/api/fdm/latest/object/networks'
```

Note This does not include the **Authorization: Bearer** header, which would be required in an API call from your client.

Request URL

The URL to issue from your client to make the request. For example, for GET /object/networks:

```
https://ftd.example.com/api/fdm/latest/object/networks
```

Response Body

The object that the system returns to your client. If a resource can include multiple objects (such as /object/network), you get a list of items on a GET request. POST/PUT/DELETE responses will be about a single object.

The specific content returned is based on the resource model. For example, GET /object/networks returns a list of objects, with each object looking something like the following (the initial indication of an items list is also shown). Note that the links/self value indicates the URL you would use to refer to this object; the object ID is included in the URL.

```
{
  "items": [
    {
      "version": "900f8558-7d19-11e7-bf7b-3dcaf0c58345",
      "name": "AIM_SERVERS-205.188.1.132",
      "description": null,
      "subType": "HOST",
      "value": "205.188.1.132",
      "isSystemDefined": true,
      "id": "900fac69-7d19-11e7-bf7b-d9417b20e59e",
      "type": "networkobject",
      "links": {
        "self": "https://ftd.example.com/api/fdm/latest/
object/networks/900fac69-7d19-11e7-bf7b-d9417b20e59e"
      }
    },
```

GET requests also include a paging section, which is explained in GET: Obtaining Data from the System, on page 37.

Response Code

The numeric HTTP status code returned for the HTTP call. These are the standard HTTP status codes, which you can find in RFCs or Wikipedia (such as https://en.wikipedia.org/wiki/List_of_HTTP_status_codes). For example, 200 (OK) indicates a successful GET/PUT/POST call, and 204 a successful DELETE call.

Response Headers

These are the packet headers in the HTTP response. For example, GET /object/networks might have headers such as the following:

```
{
  "date": "Thu, 10 Aug 2017 19:19:16 GMT",
  "content-encoding": "gzip",
  "x-content-type-options": "nosniff",
  "transfer-encoding": "chunked",
  "connection": "Keep-Alive",
  "vary": "Accept-Encoding",
  "x-xss-protection": "1; mode=block",
  "pragma": "no-cache",
  "server": "Apache",
  "x-frame-options": "SAMEORIGIN",
  "strict-transport-security": "max-age=31536000 ; includeSubDomains",
  "content-type": "application/json;charset=UTF-8",
  "cache-control": "no-cache, no-store, max-age=0, must-revalidate",
  "accept-ranges": "bytes",
  "keep-alive": "timeout=5, max=99",
  "expires": "0"
}
```

GET: Obtaining Data from the System

Use the GET method to read information from the device.

If a resource can contain multiple objects, you get a list of objects in the response. You can include query parameters on the URL to control the number of objects returned. The default is to return 10 objects from the start of the object list.

The following procedure explains the general approach for making GET calls in the API Explorer. Use the example code for your API client.

Procedure

Step 1 In API Explorer, open a GET method (first, open the group to see the methods and resources).

Step 2 If the method you want to use requires an object or parent ID in the URL, use a parent method to obtain the needed ID.

For example, GET /objects/networks/{objId} requires the ID of a specific object. Use the GET /objects/networks method to get a list of network objects, then look for the id value for the object you want to examine. Note that in this case, the information returned in the GET /object/networks call will be the same as what you see in GET /objects/network/{objId}. See Finding the Object ID (objId) and Parent ID, on page 8.

Step 3 In the **Parameters** section, configure the following options:

- **objId**—The object ID is always required if it is needed in the URL. For example, 900fac69-7d19-11e7-bf7b-d9417b20e59e.
- **parentId**—The parent ID is equivalent to object ID, but is simply for a parent that is higher up in the hierarchy. For example, GET /policy/intrusion returns a list of intrusion policies, whereas GET /policy/intrusion/{parentId}/intrusionrules returns the rules defined within one of those policies. You would get the parent ID from GET /policy/intrusion.
- **offset**—For resources that support multiple objects, how far into the list to start returning objects. The default, 0, indicates the start of the list.
- **limit**—The maximum number of objects to return in the response. The default is 10. The maximum limit is 1000; if you enter an invalid value, it is automatically changed to 1000.

- **sort**—How to sort the objects returned in the response. The default sort is alphabetical by the **name** value. To change the sort, enter the name of the attribute within the resource to sort on. For example, you can use **sort=value** in network objects to sort on the value (that is, IP address) attribute. To sort in reverse order, include a minus sign, for example, **sort=-name**.
- **filter** (not available for all resources)—Return items that match the filter criteria only. The format of the filter value is {key}{operator}{value}, where the key is an attribute name and value is the string to filter on. There are no spaces between the items. The fields you can filter on are listed in the description of the **filter** parameter in the API Explorer; the fields differ per object. If an object supports filtering on multiple fields, you can include multiple values on the **filter** parameter, separated with a semi-colon (;). For example, you could filter on gid:1;sid:105 for GET /policy/intrusionpolicies/{parentId}/intrusionrules. The allowed operators are:
    - **:** for equal to. For example, **filter=name:Canada**.
    - **!** for not equal to. For example, **filter=name!Canada**.
    - **~** for similar to. For example, **filter=name~United**.
- **filter=fts~***string* (not available for all resources)—Return items that match the filter only. The **fts~** option is for full-text search. All attributes in the object are searched for the string. You can include a partial string; using the asterisk * as a wildcard to match one or more characters is optional. Do not include the following characters, they are not supported as part of the search string: ?~!{}<>:%. The following characters are ignored: ;#&.

  For example, you could find all network objects that have 10. as the first octet using GET /object/networks?filter=fts~10. Note that it takes 3-5 seconds to index newly-created or updated objects, so you need to pause before immediately doing a full-text search on new or changed objects.
- **filter=fetchZeroHitCount:**{**true** | **false**} (available for access rules only)—If you specify **includeHitCounts=true**, you can use this filter option to either include (**true**) or exclude (**false**) rules that have not been hit, that is, whose hit count is zero. The default is **true**.
- **includeHitCounts** (available for access rules only)—Whether to include hit count information for the rules in the policy. Specify **includeHitCounts=true** to get hit counts. Specify **false** (the default) to exclude hit counts. The hit count information is returned in the hitCount attribute in the returned object.
- **time_duration** (available for trending reports only)—How many seconds into the past the report should include. For example, 1800 returns a report for the past 30 minutes.

Note Whereas {objId} and {parentId} are part of the URL path, add the **offset**, **limit**, **sort**, **filter**, **includeHitCounts**, and **time_duration** parameters after a **?** character to the end of the URL.

Step 4 Click the **Try It Out!** button and examine the response.

For successful calls (return code 200), the response body includes one object or a list of objects, depending on the call you made. For information on the general structure and content of the response, see Trying a Method and Interpreting the Results, on page 35.

GET requests include a paging section. If there are more objects than were returned for the call, the **prev** and **next** values indicate how to get the previous or next set of objects. The **count** value indicates the overall number of objects. The **limit** value indicates how many items are returned in the response. The **offset** value indicates the starting position of the returned objects, with 0 indicating the start of the list.

```
"paging": {
  "prev": [],
  "next": [
    "https://ftd.example.com/api/fdm/latest/object/networks?limit=10&offset=10"
  ],
  "limit": 10,
  "offset": 0,
  "count": 22,
  "pages": 0
}
```

POST: Create a New Object

Use the POST method to create a new object for a type of resource. For example, use POST to create a new network object.

The following procedure explains the general approach for making POST calls in the API Explorer. Use the example code for your API client.

Procedure

Step 1 In API Explorer, open a POST method (first, open the group to see the methods and resources).

Step 2 Click **Model** under the **Response Class** heading and read about the data types and values for the resource attributes.

Step 3 Under the **Parameters** heading, configure the following options if they are available:

- **parentId**—The ID of the parent object that will contain this object. For example, when adding an SSL rule, the ID of the SSL Decryption policy.
- **at**—For objects that reside in a parent that organizes the objects in an ordered list, such as SSL decryption policies, the location to insert the object. Use an integer to indicate the location, with 0 being the start of the list. The default is to insert the new object at the end of the list.

Note Whereas {objId} and {parentId} are part of the URL path, add the **at** parameter after a **?** character to the end of the URL.

Step 4 Also under the **Parameters** heading, click the JSON model shown in the **Data Type** > **Example Value** column for the **body** parameter.

Clicking in this box loads the JSON model into the Value column for the **body** parameter. For example, clicking the box for the POST /object/networks resource loads the following body:

```
{
  "name": "string",
  "description": "string",
  "subType": "HOST",
  "value": "string",
  "type": "networkobject"
}
```

Step 5 Fill in the required values for the **body** JSON object attributes.

For enum values, be certain to read the allowed values under **Response Class** > **Model**. For example, you can create a network object for a subnet (rather than a host) address by filling in the values and changing the default value for **subType**:

```
{
  "name": "new_network_object",
  "description": "A subnet object created using the REST API.",
  "subType": "NETWORK",
  "value": "10.100.10.0/24",
  "type": "networkobject"
}
```

Step 6 Click the **Try It Out!** button and examine the response.

Examine the **curl** command used to update the system. Note the additional headers. When you create your API client, you need to also include these header fields and values. For example, the **curl** command to create the sample object is the following. Note the Content-Type and Accept headers.

```
curl -X POST --header 'Content-Type: application/json' \
   --header 'Accept: application/json' -d '{ \
   "name": "new_network_object", \
   "description": "A subnet object created using the REST API.", \
   "subType": "NETWORK", \
   "value": "10.100.10.0/24", \
   "type": "networkobject" \
 }' 'https://ftd.example.com/api/fdm/latest/object/networks'
```

For successful calls (return code 200), the response body includes the complete object that you created, including other system-generated values such as **version** and **id**. The version and ID values are especially important, as you need them if you subsequently use PUT to change the object. For information on the general structure and content of the response, see Trying a Method and Interpreting the Results, on page 35.

The response body also includes the **links/self** value, which is the URL for the object you created. For example, the following is the response body for the sample object.

```
{
  "version": "f6d8da48-7ed5-11e7-9bfd-d96183b5f5f1",
  "name": "new_network_object",
  "description": "A subnet object created using the REST API.",
  "subType": "NETWORK",
  "value": "10.100.10.0/24",
  "isSystemDefined": false,
  "id": "f6d8da49-7ed5-11e7-9bfd-27136f5686ad",
  "type": "networkobject",
  "links": {
    "self": "https://ftd.example.com/api/fdm/latest/object/networks/
f6d8da49-7ed5-11e7-9bfd-27136f5686ad"
  }
}
```

PUT: Modify an Existing Object

Use the PUT method to change the attributes of an existing object. For example, use PUT to modify the address contained within an existing network object without changing the object’s ID.

The PUT method replaces the entire object. You cannot simply change one attribute. Thus, you must ensure that your JSON object includes the old values that you want to preserve.

The following procedure explains the general approach for making PUT calls in the API Explorer. Use the example code for your API client.

Before you begin

Use the GET method for the parent resource to obtain a copy of the existing state of the object, as described in GET: Obtaining Data from the System, on page 37.

You must have the correct values for at least the following parameters, as well as any user-supplied values that you do not want to change.

- **version**
- **id**

Procedure

Step 1 In API Explorer, open a PUT method (first, open the group to see the methods and resources).

Step 2 Under the **Parameters** heading, configure the following options:

- **objId**—The **id** value for the object. For example, 900fac69-7d19-11e7-bf7b-d9417b20e59e.
- **parentId**—For objects that reside within another object, the ID of the parent object that will contain this object. For example, when modifying an SSL rule, the ID of the SSL Decryption policy.
- **at**—For objects that reside in a parent that organizes the objects in an ordered list, such as SSL decryption policies, the location to insert the object. Use an integer to indicate the location, with 0 being the start of the list. The default is to insert the object at the end of the list.

Note Whereas {objId} and {parentId} are part of the URL path, add the **at** parameter after a **?** character to the end of the URL.

Step 3 Also under the **Parameters** heading, click the JSON model shown in the **Data Type** > **Example Value** column for the **body** parameter.

Clicking in this box loads the JSON model into the Value column for the **body** parameter. For example, clicking the box for the PUT /object/networks resource loads the following body. Note that this is slightly different from the POST version for the same resource: the PUT body includes the **version** attribute.

```
{
  "version": "string",
  "name": "string",
  "description": "string",
  "subType": "HOST",
  "value": "string",
  "type": "networkobject"
}
```

Step 4 Fill in the required values for the **body** JSON object attributes.

Be certain to replicate old values that you do not want to change.

For enum values, be certain to read the allowed values under **Response Class** > **Model**. Repeat the old value unless you are changing the object to a different subtype. For example, the default PUT model for a network object has HOST for **subType**, but if you are changing a subnet object, make sure you change **subType** to NETWORK.

For example, to update the subnet IP address in a network object, repeat all the old values for all attributes except **value**. Enter the new subnet address in **value**.

```
{
  "version": "f6d8da48-7ed5-11e7-9bfd-d96183b5f5f1",
  "name": "new_network_object",
  "description": "A subnet object created using the REST API.",
  "subType": "NETWORK",
  "value": "10.100.11.0/24",
  "type": "networkobject",
}
```

Step 5 Click the **Try It Out!** button and examine the response.

Examine the **curl** command used to update the system. Note the additional headers. When you create your API client, you need to also include these header fields and values. For example, the **curl** command to update the sample object is the following. Note the Content-Type and Accept headers.

```
curl -X PUT --header 'Content-Type: application/json' \
--header 'Accept: application/json' -d '{ \
   "version": "f6d8da48-7ed5-11e7-9bfd-d96183b5f5f1", \
   "name": "new_network_object", \
   "description": "A subnet object created using the REST API.", \
   "subType": "NETWORK", \
   "value": "10.100.11.0/24", \
   "type": "networkobject" \
 }' 'https://ftd.example.com/api/fdm/latest/object/
networks/f6d8da49-7ed5-11e7-9bfd-27136f5686ad'
```

For successful calls (return code 200), the response body includes the complete object that you updated. Note that the version value changes, but the object ID (and thus the link/self) stays the same. The version changes every time you modify an object. For information on the general structure and content of the response, see Trying a Method and Interpreting the Results, on page 35.

Note If you did not make any changes to the object, that is, the object being updated is the same as its previous version, the system does not process the request and instead and sends back a 204 code saying that nothing has changed for that resource.

For example, the following is the response body for updating the sample object.

```
{
  "version": "96f5f3cc-7ede-11e7-9bfd-9b7d8a92863f",
  "name": "new_network_object",
  "description": "A subnet object created using the REST API.",
  "subType": "NETWORK",
  "value": "10.100.11.0/24",
  "isSystemDefined": false,
  "id": "f6d8da49-7ed5-11e7-9bfd-27136f5686ad",
  "type": "networkobject",
  "links": {
    "self": "https://ftd.example.com/api/fdm/latest/object/networks/
f6d8da49-7ed5-11e7-9bfd-27136f5686ad"
  }
}
```

DELETE: Remove a User-Created Object

Use the DELETE method to remove an object that you, or another user, created. For example, use DELETE to remove a network object that you no longer use.

You cannot delete system-defined objects or objects that are required to exist.

You also cannot delete an object that is currently being used by another object, such as a network object that is used in an access rule. For in-use objects, first modify all objects that use it, then delete the object.

The following procedure explains the general approach for making DELETE calls in the API Explorer. Use the example code for your API client.

Before you begin

Use the GET method for the parent resource to obtain a copy of the existing state of the object, as described in GET: Obtaining Data from the System, on page 37.

You must have the object ID (the **id** value) to delete the object.

Procedure

Step 1 In API Explorer, open a DELETE method (first, open the group to see the methods and resources).

Step 2 Under the **Parameters** heading, enter the **id** value for the object in the **objId** field. For example, f6d8da49-7ed5-11e7-9bfd-27136f5686ad.

If the object resides within a container, you also need to enter the ID of the parent object into the **parentId** field.

Step 3 Click the **Try It Out!** button and examine the response.

Examine the **curl** command used to delete the object from the system. Note the additional headers. When you create your API client, you need to also include these header fields and values. For example, the **curl** command to delete the sample object is the following. Note the Accept header.

```
curl -X DELETE --header 'Accept: application/json'
'https://ftd.example.com/api/fdm/latest/object/
networks/f6d8da49-7ed5-11e7-9bfd-27136f5686ad'
```

For successful calls (return code 204 “No Content”), you get an empty response body. This is the expected result.

CHAPTER 7

Deploying Configuration Changes

Deploying Configuration Changes

Although POST, PUT, and DELETE calls directly update the threat defense device, they are not immediately active. You must deploy configuration changes before the device uses your new settings when processing traffic.

Procedure

Step 1 Use the POST /operational/deploy resource in the Deployment group to initiate a deployment.

For example, the **curl** command would look like the following:

```
curl -X POST --header 'Content-Type: application/json'
--header 'Accept: application/json'
'https://ftd.example.com/api/fdm/latest/operational/deploy'
```

Step 2 Evaluate the response to verify that the deployment job was queued.

A good response (status code 200) looks like the following. Note the state.

```
{
  "id": "62bf405f-796c-11e8-8640-a9156b92ec49",
  "statusMessage": null,
  "statusMessages": null,
  "modifiedObjects": {},
  "cliErrorMessage": null,
  "queuedTime": 1530036705491,
  "startTime": -1,
  "endTime": -1,
  "state": "QUEUED",
  "name": "User (admin) Triggered Deployment",
  "links": {
    "self": "https://ftd.example.com/api/fdm/latest/operational/deploy/
62bf405f-796c-11e8-8640-a9156b92ec49"
  }
}
```

Note The **cliErrorMessage** and **name** attributes were added in API v2; they are not included in v1 responses.

Step 3 Use the GET /operational/deploy/{objId} resource to check the status of the job.

For example, the **curl** command would look like the following:

```
curl -X GET --header 'Accept: application/json'
'https://ftd.example.com/api/fdm/latest/operational/deploy/
a7a227fb-82ab-11e7-8186-0dc471ff0672'
```

The response might look like the following. Note the state, DEPLOYED, indicates the job completed successfully. The modifiedObjects parameter lists the objects that were changed in the deployment job. In this case, there is a single change, to a network object named new-network.

```
{
  "id": "62bf405f-796c-11e8-8640-a9156b92ec49",
  "statusMessage": "Deployed Successfully",
  "statusMessages": [
    "Deployed Successfully"
  ],
  "modifiedObjects": {
    "NetworkObject": [
      "new-network"
    ]
  },
  "cliErrorMessage": null,
  "queuedTime": 1530036705491,
  "startTime": 1530036705924,
  "endTime": 1530036822612,
  "state": "DEPLOYED",
  "name": "User (admin) Triggered Deployment",
  "links": {
    "self": "https://ftd.example.com/api/fdm/latest/operational/deploy/
62bf405f-796c-11e8-8640-a9156b92ec49"
  }
}
```

CHAPTER 8

Configuration Import/Export

Version Requirement: To use configuration import/export, you must be running the threat defense version 6.5(0) or higher, and the threat defense REST API v4 or higher.

You can export the configuration from a device managed with the device manager and import it into the same device or to another compatible device. For example, you can use configuration import/export to replicate a baseline configuration across multiple similar devices, then use the device manager on each device to configure the characteristics unique to each device.

About Configuration Import/Export

When you manage the threat defense device locally, with the device manager or through the CDO, you can export the configuration of the device using the threat defense API. This method does not work with a device managed by the Secure Firewall Management Center.

When you export the configuration, the system creates a zip file. You can then download the zip file to your workstation. The configuration itself is represented as objects defined using attribute-value pairs in a JSON-formatted text file. You can edit the file prior to importing it back into the same device or a different device.

Thus, you can use an export file to create a template that you can deploy to other devices in your network.

When importing objects, you also have the option of defining the objects directly in the import command rather than in a configuration file. However, you should directly define objects only in cases where you are importing a small number of changes.

The following topics explain more about configuration import/export.

What is Included in the Export File

When you do an export, you specify which configurations to include in the export file. A full export includes everything in the export zip file. Based on what you choose to export, the export zip file might include the following:

- Attribute-value pairs that define each configured object. All configurable items are modeled as objects, not just those that are called “objects” in the device manager.

- If you configured remote access VPN, the AnyConnect packages and any other referenced files, such as client profile XML files, the DAP XML file, and Hostscan packages.
- If you configured custom file policies, any referenced clean list or custom detection list.

Comparing Import/Export and Backup/Restore

Configuration import/export is not the same as backup/restore.

- Backup/restore is for disaster recovery. You can restore a backup to a device only if the device is the same model, and running the same software version, as the device from which the backup was taken. Primarily, this is for recovering the “last good” configuration to the same device, or to restore the configuration to a replacement device.
- Import/export is for preserving all or part of a configuration. You can use an export file to restore the configuration to a device after you reimage it. Or, you can use the export file as a template, editing the contents before importing it into another device. With import/export, you can quickly get a new device up to a certain baseline configuration, so you can deploy it more rapidly into your network. Within limits, you can even import a file to different device models, for example, from a Firepower 2120 to a 2130. If the import file only includes objects that are supported on all device models, there should be very few restrictions on import. The one restriction is that the device needs to use the same API version used for the export file.

Strategies for Import/Export

Following are some ways you can use import/export.

- **Create a template for new devices.** Configure your model device to the baseline you need, then export the full configuration. Subsequently, you can import that configuration into new devices, then use the device manager or the threat defense API to make whatever modifications are needed. You can also edit the template prior to import to make these modifications, for example, to the IP addresses for each interface. Note that the full export includes the ManagementIP object (type=managementip); assuming that you have already configured the management address and gateway on the target device, you should remove this object from the export file when you create the template for the new device, or you will overwrite the management addressing information.
- **Deploy configuration changes from one device to other similar devices.** For example, when editing the configuration of device A, you create a few new network objects and access control rules. You can then export the pending changes, and import those changes into device B. After you deploy the configuration on both devices, they are running the same new rules.
- **Reapply the configuration after a system reimage.** Reimaging a device erases the configuration. If you first export the full configuration, you can them import it after you complete the reimage.
- **Apply targeted configurations.** Because you can edit or even manually create an export file, you can remove all objects except those you want to import into another device. For example, you could create a configuration file that contains a set of network objects, and use it to import the same group of network objects into all of your threat defense devices.

Guidelines for Configuration Import/Export

- During an export job, the system holds a write lock on the configuration database. You cannot use the API, or the device manager, to make configuration changes until the job completes. However, you can view the configuration in the device manager, or use GET calls in the API, during the export job.
- During an import job, the system holds both read and write locks on the configuration database. You cannot use the API or the device manager to view the configuration or make changes to it until the job completes.
- The imported configuration is added to the existing configuration. You cannot wipe away the device's configuration and replace it with the imported configuration. If you need to reset the device configuration prior to import, you can go to the device CLI and issue the **configure manager delete** command, followed by the **configure manager local** command. Only the management interface configuration will be preserved.
- You can import a file into a device only if the device is running the same API version as defined in the apiVersion attribute in the metadata object contained in the file.
- The SRU version must be the same on the export and import devices, or the import will fail.

Importing and Exporting Configurations

The import/export process starts with exporting the configuration from a locally-managed device. You can then download the export file, and optionally edit it, before uploading it into the same device or a compatible device. The following topics explain each step.

Export the Configuration

Use the POST /action/configexport method to create and start a configuration export job.

Procedure

Step 1 Create the JSON object body for the export job.

Following is an example of the JSON object to use with this call.

```
{
  "diskFileName": "string",
  "encryptionKey": "*********",
  "doNotEncrypt": false,
  "configExportType": "FULL_EXPORT",
  "deployedObjectsOnly": true,
  "entityIds": [
    "string"
  ],
  "jobName": "string",
  "type": "scheduleconfigexport"
}
```

The attributes are:

- **diskFileName**—(Optional.) The name of the export zip file. If you do not specify a name, the system generates one for you. Even if you specify a name, the system might append characters to the name to ensure uniqueness. The name has a maximum length of 60 characters.
- **encryptionKey**—(Optional.) An encryption key for the zip file. If you do not want to encrypt the file, omit this field and specify "doNotEncrypt": true instead. If you specify a key, you will need to use the key to open the zip file after you download it to your workstation. Note that the exported configuration file exposes secret keys, passwords, and other sensitive data in clear text (because otherwise they cannot be imported), so you might want to apply an encryption key to protect sensitive data. The system uses AES 256 encryption.
- **doNotEncrypt**—(Optional.) Whether the export file should be encrypted (false), or not encrypted (true). The default is false, which means you must specify a non-empty encryptionKey attribute. If you specify true, then the encryptionKey attribute is ignored.
- **configExportType**—One of the following enum values:
    - **FULL_EXPORT**—Include the entire configuration in the export file. This is the default.
    - **PARTIAL_EXPORT**—Include only those objects, and their descendant objects, that are identified in the entityIds list. Unexportable objects are not included even if you specify their identities. All user-defined objects are exportable.
    - **PENDING_CHANGE_EXPORT**—Include only those objects that have not yet been deployed, that is, the pending changes.
- **deployedObjectsOnly**—(Optional.) Whether to include objects in the export file only if they have been deployed. That is, do not include pending changes. This attribute is ignored for PENDING_CHANGE_EXPORT jobs, because those jobs include undeployed objects only. The default is false, which means all pending changes are included in the export. Specify true to exclude pending changes.
- **entityIds**—A comma-separated list of the identities of a set of starting-point objects, enclosed in [brackets]. This list is required for a PARTIAL_EXPORT job. Each item in this list could be either a UUID value or an attribute-value pair matching patterns like "**id=***uuid-value*", "**type**=*object-type*" or "**name**=*object-name*". For example, **"type=networkobject"**.

    The **type** can be either a leaf entity, such as networkobject, or an alias of a set of leaf types. Some typical type aliases are: network (NetworkObject and NetworkObjectGroup), port (all TCP/UDP/ICMP port, protocol and group types), url (URL objects and groups), ikepolicy (IKE V1/V2 policies), ikeproposal (Ike V1/V2 proposals), identitysource (all identity sources), certificate (all certificate types), object (all object/group types that would be listed in the device manager on the Objects page), interface (all network interfaces, s2svpn (all site-to-site VPN related types), ravpn (all RA VPN related types), vpn (both s2svpn and ravpn).

    All of these objects and their outgoing referential descendants will be included in the PARTIAL_EXPORT output file. Note all the unexportable objects will be excluded from the output even if you specify their identities. Use the GET method for the appropriate resource types to obtain the UUIDs, types, or names for the target objects.

    For example, to export all network objects, plus an access rule named myaccessrule, and two objects identified by UUID, you can specify:

    ```
    "entityIds": [
    ```

```
        "type=networkobject",
        "id=bab3e3cd-8c70-11e9-930a-1f12ee87d473",
        "name=myaccessrule",
        "acc2e3cd-8c70-11e9-930a-1f12ee87b286"
        ]",
```

- **jobName**—(Optional.) A name for the export job. Giving the job a name might make it easier to find it when you retrieve job status.
- **type**—The job type, which is always **scheduleconfigexport**.

Example:

The following example performs a full export to the file export-config-1 and accepts the defaults for all other attributes:

```
{
  "diskFileName": "export-config-1",
  "doNotEncrypt": true
  "configExportType": "FULL_EXPORT",
  "type": "scheduleconfigexport"
}
```

Step 2 Post the object.

For example, the curl command would look like the following:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json'
 -d '{ \
   "configExportType": "FULL_EXPORT", \
   "type": "scheduleconfigexport" \
 }' 'https://10.89.5.38/api/fdm/latest/action/configexport'
```

Step 3 Verify the response.

You should get a response code of 200. A successful response body would look something like the following if you posted the minimum JSON object. If you specify an encryption key, it is masked in the response.

```
{
  "version": null,
  "scheduleType": "IMMEDIATE",
  "user": "admin",
  "forceOperation": false,
  "jobHistoryUuid": "c7a8ba61-629a-11e9-8b8d-0fcc3c9d6d0b",
  "ipAddress": "10.24.5.177",
  "diskFileName": "export-config-1",
  "encryptionKey": null,
  "doNotEncrypt": true
  "configExportType": "FULL_EXPORT",
  "deployedObjectsOnly": false,
  "entityIds": null,
  "jobName": "Config Export",
  "id": "c79be920-629a-11e9-8b8d-85231be77de0",
  "type": "scheduleconfigexport",
  "links": {
    "self": "https://10.89.5.38/api/fdm/latest
/action/configexport/c79be920-629a-11e9-8b8d-85231be77de0"
  }
}
```

Check the Status of the Export Job

It takes some time for an export job to complete. The larger the configuration, the more time the job will require. Check the job status to ensure it completes successfully before you try to download the file.

The simplest way to get status is to use GET /jobs/configexportstatus. For example, the curl command would look like the following:

```
curl -X GET --header 'Accept: application/json'
'https://10.89.5.38/api/fdm/latest/jobs/configexportstatus'
```

A successfully completed job would return status similar to the following.

```
{
  "version": "hdy62yf5xp3vf",
  "jobName": "Config Export",
  "jobDescription": null,
  "user": "admin",
  "startDateTime": "2019-04-19 13:14:54Z",
  "endDateTime": "2019-04-19 13:14:56Z",
  "status": "SUCCESS",
  "statusMessage": "The configuration was exported successfully",
  "scheduleUuid": "1ef502ad-62a5-11e9-8b8d-074ebc750708",
  "diskFileName": "export-config-1.zip",
  "messages": [],
  "configExportType": "FULL_EXPORT",
  "deployedObjectsOnly": false,
  "entityIds": null,
  "id": "1f0aad8e-62a5-11e9-8b8d-bb1ebb4d1300",
  "type": "configexportjobstatus",
  "links": {
    "self": "https://10.89.5.38/api/fdm/latest
/jobs/configexportstatus/1f0aad8e-62a5-11e9-8b8d-bb1ebb4d1300"
  }
}
```

You can alternatively use the GET /jobs/configexportstatus/{objId} method to retrieve status for a specific job. You would get the object ID from the **id** field in the response object.

Download the Export File

When an export job completes, the export file is written to the system disk and is called a configuration file. You can download this export file to your workstation using the GET /action/downloadconfigfile/{objId} method. To get a list of the available files, use the GET /action/configfiles method.

Note With GET /action/downloadconfigfile/{objId} you typically specify the file name as the object ID. Alternatively, you can specify the ID of the ConfigExportStatus object associated with the file.

Procedure

Step 1 Get a list of the configuration files on the disk.

The list of configuration files includes export files and any files that you uploaded for import.

The curl command would be similar to the following:

```
curl -X GET --header 'Accept: application/json'
'https://10.89.5.38/api/fdm/latest/action/configfiles'
```

The response would show a list of items, each of which is a configuration file. For example, the following list shows 2 files. Note that the **id** for all files is default. Ignore the ID, and use the **diskFileName** instead.

```
{
  "items": [
    {
      "diskFileName": "export-config-2.zip",
      "dateModified": "2019-04-19 13:32:28Z",
      "sizeBytes": 10182,
      "id": "default",
      "type": "configimportexportfileinfo",
      "links": {
        "self": "https://10.89.5.38/api/fdm/latest/action/configfiles/default"
      }
    },
    {
      "diskFileName": "export-config-1.zip",
      "dateModified": "2019-04-19 13:14:56Z",
      "sizeBytes": 10083,
      "id": "default",
      "type": "configimportexportfileinfo",
      "links": {
        "self": "https://10.89.5.38/api/fdm/latest/action/configfiles/default"
      }
    }
  ],
```

Step 2 Download the file using the diskFileName as the object ID.

The curl command would be similar to the following:

```
curl -X GET --header 'Accept: application/octet-stream'
'https://10.89.5.38/api/fdm/latest/action/downloadconfigfile/export-config-2.zip'
```

The file is downloaded to your default downloads folder. If you are issuing the GET method from the API Explorer, and your browser is configured to prompt for download location, you will be prompted to save the file.

A successful download will result in a 200 return code and no response body.

Edit the Exported Configuration File

After you download the configuration file, you can unzip it and open the text file that contains the objects. WordPad formats the content in an easier to read fashion than NotePad. You can also use other text editors that you might have installed. You can even create your own configuration file from scratch, but you will need to export the configuration to understand the file structure.

The following topics explain the requirements for the text file.

Minimum Configuration File Requirements

A configuration file must have the following minimum elements:

- Enclose the objects in the file within [brackets]. The entire file uses standard JSON notation and is an array of objects.
- Enclose each object in {braces}.
- Use commas to separate the objects in the configuration file. That is, the end brace of an object should be followed by a comma except for the final object.
- The first object in the file must be a metadata object. The easiest way to get the right object attributes is to export the configuration from a device of the desired model. For example, following is the metadata object from a Secure Firewall Threat Defense Virtual device. Before importing the device, you can edit the configuration and export types, and if desired, delete the generatedOn attribute.

```
{"hardwareModel":"Cisco Firepower Threat Defense for VMWare",
 "type":"metadata",
 "configType":"FULL_CONFIG",
 "apiVersion":"latest",
 "generatedOn":"Fri Apr 19 13:32:28 UTC 2019",
 "exportType":"FULL_EXPORT",
 "softwareVersion":"6.5.0-10480"}
```

- The metadata object must specify the appropriate configuration type (configType) value.
    - FULL_CONFIG—This text file includes the full device configuration.
    - DELTA_CONFIG—This text file includes a partial configuration, perhaps even just a few objects.
- The exportType is one of the following: FULL_EXPORT, PARTIAL_EXPORT, PENDING_CHANGE_EXPORT.
- If you are doing a full configuration import, the metadata object must specify the following attributes: hardwareModel, softwareVersion, apiVersion.
- You can write objects on one line or on multiple lines, but do not put empty lines or comment lines between the attributes in an object. Comments are not allowed in the file.
- Although objects are exported in dependency order, where an object referred to by another object is defined first, maintaining that order in an import configuration file is not required. The system will automatically resolve relationships during import, assuming the object names and IDs resolve correctly between the dependent objects.

Basic Structure of Identity Wrapper Objects

The configuration file uses identity wrapper objects to define any ConfigEntity or ManagementEntity object that can be exported or imported. Following is the basic structure of an identity wrapper object:

```
{
    "type" : "identitywrapper",
    "data" : {},
    "parentName" : "container-name",
    "oldName" : "old-object-name",
    "action" : "EDIT", //Enum values: CREATE, EDIT or DELETE
    "index" : integer,
  }
```

The object contains the following attributes:

- **type**—This is always **identitywrapper**.
- **data**—This is the collection of attribute-value pairs that define the object from the configuration, such as a network object, access control rule, and so forth. The attributes needed in this collection depend on the model for the specific object type and the action you are taking. Enclose the attribute-value pairs in {braces}. Separate the attributes within the data array with commas.
- **parentName**—(If needed.) A limited number of objects are ContainedObjects, which have a relationship to an object that contains them. Examples include access rules, manual NAT rules, and subinterfaces. For these items, the parentName specifies the name of the containing object (the parent). Specify this attribute for contained objects. Do not specify it for non-contained objects. You might also need to specify index for these objects.

  You can actually omit this attribute if the parent is a single object (that is, you cannot create more than one), such as AccessPolicy, and the system can resolve the reference.
- **oldName**—(If needed.) If you are renaming an existing object, you can specify the old name on this attribute, and the new name in the **name** attribute of the data attributes. The action must be EDIT to use this attribute.
- **action**—The action to take with respect to the defined object. In full exports, the action is always **CREATE**. For pending change or partial exports, other actions might be **EDIT** or **DELETE**.

  When you edit the file for import, specify the desired action. Note that if you specify CREATE but the object already exists, the action is changed to EDIT; if the object does not exist, EDIT is changed to CREATE. The DELETE action is not changed. Object references are resolved based on object type and name, or object type and old name, or object type and parent name.

  - CREATE—This is a new object. You need to specify the data attributes that are required when posting an object. Note that if the **name** matches an existing object of the specified type, the action is automatically changed to EDIT.

    Note that if you create a new object and reference that object from other objects, such as defining a network object and then using it in an access rule, the object **name** must be correct in the reference.
  - EDIT—You are updating an object. You need to specify the data attributes that are required when putting an object, except for version and id. The name and object type are used to determine the object to update, and the version attribute is always ignored.
  - DELETE—You are deleting the object. You must specify the **type** and **name** attributes in the object data.
- **index**—(Optional; integer.) For objects that are part of an ordered list, such as access control and manual NAT rules, the position of the object in the policy. If you are creating a new rule and you do not specify an index value, the rule is added to the end of policy as the last rule. If you are editing the rule, the system will retain the rule's existing position.

Example: Editing a Network Object for Import Into a Different Device

Each object is structured like the following, which is a network host object that defines the IP address of the syslog server:

```
{"type":"identitywrapper",
 "action":"CREATE",
 "data":{
    "version":"lfxdbtbyg4ex6",
    "name":"syslog-host",
    "subType":"HOST",
    "value":"10.100.10.10",
    "isSystemDefined":false,
    "dnsResolution":"IPV4_AND_IPV6",
    "id":"2cd0ea03-62a7-11e9-8b8d-dbf377c781d8",
    "type":"networkobject"}}
```

Suppose you exported this object from a device, and you want to import the object into a different device, but the new device should use a syslog server at a different address, 192.168.5.15. Because you are going to create a new object, remove the **version** and **id** attributes from the data attribute. You can also remove **isSystemDefined** (whose default is false) and **dnsResolution** (which is relevant for an FQDN object only). The resulting new object would look like the following:

```
{"type":"identitywrapper",
 "action":"CREATE",
 "data":{
    "name":"syslog-host",
    "subType":"HOST",
    "value":"192.168.5.15",
    "type":"networkobject"}}
```

At the top of the file, you need to retain (or add) the metadata object. You can also add line returns to make it easier to scan and verify the file content. Thus, the complete configuration file would look like the following:

```
[
{"hardwareModel":"Cisco Firepower Threat Defense for VMWare",
 "type":"metadata",
 "configType":"DELTA_CONFIG",
 "apiVersion":"latest",
 "exportType":"PARTIAL_EXPORT",
 "softwareVersion":"6.5.0-10465"}
,
{"type":"identitywrapper",
 "action":"CREATE",
 "data":{
    "name":"syslog-host",
    "subType":"HOST",
    "value":"192.168.5.15",
    "type":"networkobject"}}
]
```

Upload the Import File

Before you can import a configuration file into a device, you must first upload the file to the device. You can upload either zip or text files. You can include AnyConnect packages and client profiles if you use a zip file.

Use the POST /action/uploadconfigfile resource to upload the file. The name has a maximum length of 60 characters.

- If you use this method from API Explorer, click the **Choose File** button next to the **fileToUpload** attribute to select the file from your workstation drive.

- If you are using the method from your own program, the request payload must contain a single file-item with a file-name field. The file-name extension must be either .txt or .zip and the actual file content format must be consistent with the file extension.

The curl command would look like the following:

```
curl -F 'fileToUpload=@./import-1.txt'
'https://10.89.5.38/api/fdm/latest/action/uploadconfigfile'
```

A successful transfer results in a 200 return code and a response body similar to the following, which shows the file name on the threat defense system (**diskFileName**), which you need for the import job.

```
{
  "diskFileName": "import-1.txt",
  "dateModified": "2019-04-22 10:18:12Z",
  "sizeBytes": 267,
  "id": "default",
  "type": "configimportexportfileinfo",
  "links": {
    "self": "https://10.89.5.38/api/fdm/latest/action/uploadconfigfile/default"
  }
}
```

Import the Configuration and Check Job Status

After you upload a configuration file to the threat defense system, you can import the objects defined in the configuration file into the threat defense configuration. Use the POST /action/configimport method.

When importing objects, you also have the option of defining the objects directly in the import command rather than in a configuration file. However, you should directly define objects only in cases where you are importing a small number of changes, such as one or two network objects.

Procedure

Step 1 Create the JSON object body for the import job.

Following is an example of the JSON object to use with this call.

```
{
  "diskFileName": "string",
  "encryptionKey": "*********",
  "preserveConfigFile": true,
  "autoDeploy": true,
  "allowPendingChange": true,
  "excludeEntities": [
     "string"
     ]",
  "inputEntities": [
    {
      "action": "CREATE",
      "oldName": "string",
      "parentId": "string",
      "parentName": "string",
      "index": 0,
      "data": {
        "version": "string",
        "id": "string",
        "type": "identity"
      },
      "id": "string",
      "type": "IdEntityWrapper"
    }
  ],
  "jobName": "string",
  "type": "scheduleconfigimport"
}
```

The attributes are:

- **diskFileName**—The name of the configuration zip or txt file to be imported.
- **encryptionKey**—The key used to encrypt the zip file, if any. Do not specify a key if the configuration file is not encrypted.
- **preserveConfigFile**—(Optional.) Whether to keep the copy of the configuration file imported on the threat defense disk after a successful import job. Specify true to keep the file, false to have the file deleted from the threat defense disk. The default is false.
- **autoDeploy**—(Optional.) Whether to automatically start a deployment job if the import is successful. Imported objects are pending changes, and they are not active until you successfully deploy the changes. Specify true to start the deployment job automatically. If you specify false, you must manually deploy your changes. The default is false.
- **allowPendingChange**—(Optional.) Whether to allow the import job to start if there are existing pending changes. If you set this attribute to true, and autoDeploy to true, then the automatic deployment job includes all changes, both pre-existing and imported. If you set this attribute to false, then the import job will not run if there are pending changes. The default is false.
- **excludeEntities**—(Optional.) A list of object matching strings that identify objects that should not be imported. You need to specify this attribute only if the import file includes items that you do not want to import (that is, you decided to not delete them from the file you uploaded). Each item in this list has a pattern like "**id=***uuid-value*", "**type**=*object-type*" or "**name**=*object-name*". Input objects that match one of these patterns will be excluded from import.

  The **type** can be either a leaf entity, such as networkobject, or an alias of a set of leaf types. Some typical type aliases are: network (NetworkObject and NetworkObjectGroup), port (all TCP/UDP/ICMP port, protocol and group types), url (URL objects and groups), ikepolicy (IKE V1/V2 policies), ikeproposal (Ike V1/V2 proposals), identitysource (all identity sources), certificate (all certificate types), object (all object/group types that would be listed in the device manager on the Objects page), interface (all network interfaces, s2svpn (all site-to-site VPN related types), ravpn (all RA VPN related types), vpn (both s2svpn and ravpn).

  For example, to exclude all network objects, and two other objects identified by the name myobj and a UUID from being imported, specify:

  ```
  "excludeEntities": [
     "type=networkobject",
     "name=myobj",
     "id=acc2e3cd-8c70-11e9-930a-1f12ee87b286"
     ]",
  ```

- **inputEntities**—If you have a small number of objects to import, you can define them in the inputEntities object list rather than in a configuration file. To use this attribute, you cannot include the diskFileName attribute, or you must set that attribute to null.

- **jobName**—(Optional.) A name for the export job. Giving the job a name might make it easier to find it when you retrieve job status.
- **type**—The job type, which is always **scheduleconfigimport**.

Example:

The following example imports the configuration file named import-1.txt:

```
{
  "diskFileName": "import-2.txt",
  "preserveConfigFile": true,
  "autoDeploy": true,
  "allowPendingChange": true,
  "type": "scheduleconfigimport"
}
```

Step 2 Post the object.

For example, the curl command would look like the following:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json'
 -d '{ \
   "diskFileName": "import-2.txt", \
   "preserveConfigFile": true, \
   "autoDeploy": true, \
   "allowPendingChange": true, \
   "type": "scheduleconfigimport" \
 }' 'https://10.89.5.38/api/fdm/latest/action/configimport'
```

Step 3 Verify the response.

You should get a response code of 200. A successful response body would look something like the following if you posted the minimum JSON object. If you specify an encryption key, it is masked in the response.

```
{
  "version": null,
  "scheduleType": "IMMEDIATE",
  "user": "admin",
  "forceOperation": false,
  "jobHistoryUuid": "7e360139-6725-11e9-abb5-078014531401",
  "ipAddress": "10.24.127.37",
  "diskFileName": "import-2.txt",
  "encryptionKey": null,
  "preserveConfigFile": true,
  "autoDeploy": true,
  "allowPendingChange": true,
  "jobName": "Config Import",
  "id": "7e2b52d8-6725-11e9-abb5-5dec35337506",
  "type": "scheduleconfigimport",
  "links": {
    "self": "https://10.89.5.38/api/fdm/latest
/action/configimport/7e2b52d8-6725-11e9-abb5-5dec35337506"
  }
}
```

Step 4 Use GET /jobs/configimportstatus to check the status of the import job.

Alternatively, you can use GET /jobs/configimportstatus/{objId} to get status of one import job. For objId, use the jobHistoryUuid value from the response body to your POST /action/configimport call.

The curl command would look like the following:

```
curl -X GET --header 'Accept: application/json'
'https://10.89.5.38/api/fdm/latest/jobs/configimportstatus'
```

The response body might look like the following for a successful import. If the import fails, you might need to edit the file to correct formatting or content errors and try again.

```
{
      "version": "pcgccfnk4hmiz",
      "jobName": "Config Import",
      "jobDescription": null,
      "user": "admin",
      "startDateTime": "2019-04-25 06:43:54Z",
      "endDateTime": "2019-04-25 06:44:01Z",
      "status": "SUCCESS",
      "statusMessage": "The configuration was imported successfully",
      "scheduleUuid": "7e2b52d8-6725-11e9-abb5-5dec35337506",
      "diskFileName": "import-2.txt",
      "messages": [],
      "preserveConfigFile": true,
      "autoDeploy": true,
      "allowPendingChange": true,
      "id": "7e360139-6725-11e9-abb5-078014531401",
      "type": "configimportjobstatus",
      "links": {
        "self": "https://10.89.5.38/api/fdm/latest
/jobs/configimportstatus/7e360139-6725-11e9-abb5-078014531401"
  }
}
```

What to do next

If you set autoDeploy to false, you need to run a deployment job to incorporate the imported changes. Use the POST /operational/deploy method. If you set it to true, the configuration should have been deployed successfully. In the device manager or the API (GET /operational/auditevents), you can check the audit log, and the deployment job is named “Post Configuration Import Deployment.”

Note Some features require particular licenses. For example, a device must have a license for any remote access VPN features. However, the import process does not validate licenses. Thus, if you import objects for a license-controlled feature to a device that does not have the required license, the deployment job will fail. If you encounter this problem, either assign the required licenses to the device, or delete the objects.

Delete Unneeded Import/Export Files

If you no longer need a configuration file, either one created by an export job or one that you uploaded for configuration import, you can delete the file.

Use the DELETE /action/configfiles/{objId} method, using the file name as the objId value.

For example, to delete the file named export-config-2.zip, the curl command would be the following:

```
curl -X DELETE --header 'Accept: application/json'
'https://10.89.5.38/api/fdm/latest/action/configfiles/export-config-2.zip'
```

A successful result is a 204 return code with no response body.

You can use GET /action/configfiles to confirm that the file was deleted.

CHAPTER 9

For More Information and Examples

For More Information and Examples

You can find additional information on how to use the API from the following sites:

- https://developer.cisco.com/site/ftd-api-reference/

  This site includes reference information for the resources, including examples for Bash calls and Python code. There is a menu to select which version of the API you are using. Please select the appropriate version to see correct reference information. There is also a list of all of the unique error codes and messages that you might see when using the API.

- https://developer.cisco.com/docs/firepower/threat-defense/

  This site has end-to-end examples for configuring select features, such as high availability, including code samples.

- https://developer.cisco.com/firepower/threat-defense/

  This site includes videos, learning modules, and labs to help you learn how to use the API.