SOPHOS

Security made simple.

Sophos UTM

RESTful API

Document date: Tuesday, September 19, 2017

Limited Warranty

Contents

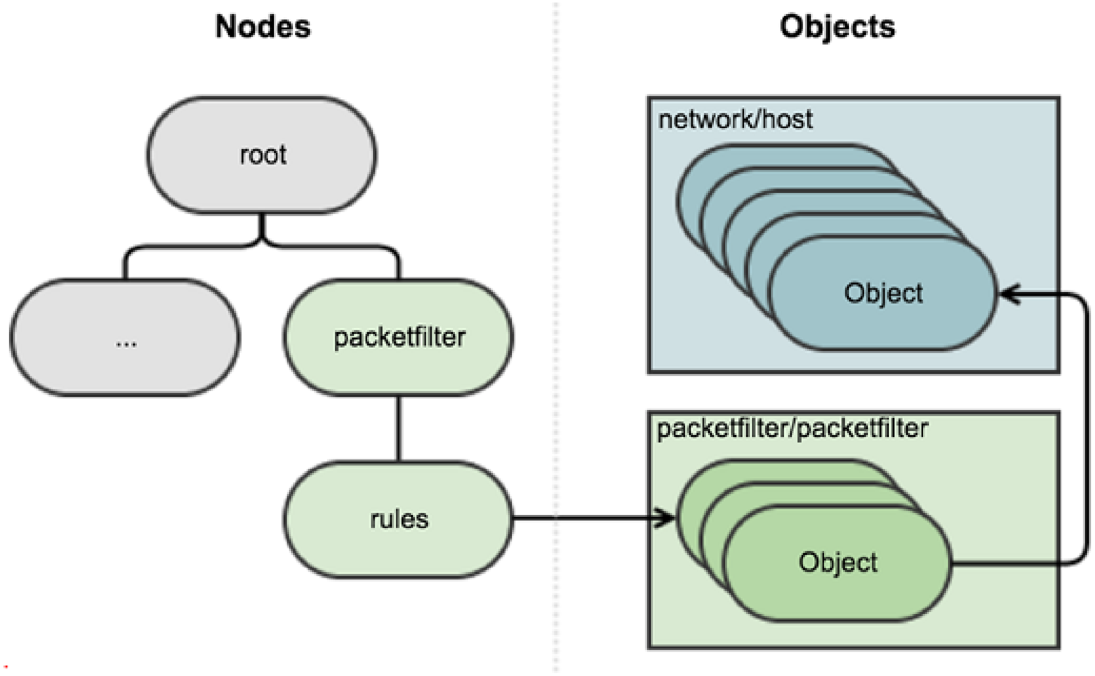
1 Sophos UTM RESTful API

Sophos UTM provides a Representational State Transfer (REST) Application Programming Interface (API) as a way of interacting with external systems. In general, systems that support RESTful API calls allow services or software to access or change system resources using a predefined set of operations.

Sophos UTM supports RESTful API calls that allow you to programmatically update policies, make configuration changes, or automatically provision a system with a default set of rules. The purpose of the RESTful API on Sophos UTM is to help you automate tasks in order to ensure your security rules are consistent and avoid human error where possible.

For more information on RESTful APIs in general, see Wikipedia.

Sophos UTM uses a configuration management service called confd which allows users to make configuration changes. The RESTful API service (called restd) enables direct access to confd which allows API calls to Sophos UTM. Confd is comprised of nodes and objects which are organized in a pre-defined hierarchy and collections respectively.



Nodes and objects

1.1 Nodes

Nodes are resources within Sophos UTM that you can update but cannot create, delete, or move. Nodes are organized hierarchically as in a filesystem. An example of a node is the “Shell Access” service module within Sophos UTM (identified as the “ssh” node). To enable the *Shell Access* service, you can set the `ssh.status` leaf node.

Unlike filesystems which use a slash “/” to separate the different nodes, Sophos UTM RESTful API uses a dot “.” to separate different nodes. Nodes reference objects, for example “Shell Access” and have a leaf node of `allowed_networks` which is an array of references to network objects.

1.2 Objects

Objects reside in collections, which you can create, change, or delete. The collections of objects are predefined into classes and types. A class describes the general purpose of the objects whereas the type describes the required data for the object. Some objects need to be referenced by a node, to work properly (e.g. packetfilter/packetfilter and packetfilter/nat).

For example, one of most used class is “network”. The network class describes objects like “host” for real hosts (e.g. 192.168.0.1) or “network” for subnets (e.g. 192.168.0.0/24). The collection of objects is always expressed with a class and a type, e.g. “network/host” or “network/network”.

1.3 References

References are the connections between nodes and objects as well as between one object and another object. Each confd node and object has a list of attributes with predefined types, where one of the types can be a reference. Please note however that you can’t create a reference in all cases. You can only make a reference in scenarios where nodes and objects are designed to be connected. Technically, references are strings that always start with the prefix “REF_”.

1.4 Validation

Confd validates all nodes and objects on change operations (e.g., create, update, delete). When you execute a RESTful API call, you may trigger some validation errors.

Validation errors are:

- fatal
- non-fatal

Fatal errors indicate a programming error like wrong input or wrong type. Non-fatal errors indicate that the current operation has impact on other objects or nodes, which might be harmful.

For example, if you create a rule for “packetfilter/packetfilter”, you would need to reference the node “packetfiler/rules” and enable the status. If you deleted the rule afterwards, confd would detect that a referenced rule of “packetfilter/rules” was still enabled and report an error.

For more information on validation errors, see chapter *X-Restd-Err-Ack header*.

2 Preparation

Before enabling the RESTful API, make sure you have performed a basic Sophos UTM setup and you have set parameters such as authentication and authorization.

2.1 Basic Setup

For information and how to perform the Basic Setup in your environment, see the Sophos UTM Quick Start Guide.

You can skip this step if you're using Sophos UTM on AWS with AWS CloudFormation as the CloudFormation templates preform the Basic Setup.

2.2 Enabling the RESTful API

Before making RESTful API calls to Sophos UTM, ensure that the service is enabled and running. You can skip this step in AWS deployments because the RESTful API service is enabled by default.

1. Login to the WebAdmin GUI.
2. Go to *Management* > *WebAdmin Settings* > *RESTful API*.
3. Activate *Enable RESTful API*.

Alternatively, you can enable the RESTful API by using the `cc` command inside a secure shell:

1. Login to Sophos UTM via SSH.

   Note – For information on how to do this, see the UTM 9.x Administration Guides.

2. Execute the command `utm:/root # cc set webadmin rest_api 1`.

2.3 Authenticating to Sophos UTM

In order to make secure RESTful API calls to Sophos UTM you will have to configure proper authentication to Sophos UTM (in general) and proper authentication for issuing API calls.

To securely access Sophos UTM you can use the SSL certificate that is auto generated by Sophos UTM. In most environments, the certificate is self-signed and not part of the certificate validation change. You can follow the steps listed in Create and Import a Public Signed Certificate for UTM Web Application Security for creating a SSL certificate that is authorized by a Certificate Authority (CA).

2.4 Authenticating for API calls

You can authenticate API calls using different methods:

- Username and password credentials
- API tokens

2.4.1 Username and password credentials

Using username and password credentials, you can make API calls with a user account that has access to Sophos UTM. Although the information is encrypted using HTTPS, we do not recommend using username and password credentials for API calls because revoking API access means identifying the user account and disabling/changing credentials.

2.4.2 API tokens

If you decide to use an API Token to authenticate, you need to create a token that can be used for making individual API calls. To set up the token, perform the following steps:

1. Login to the WebAdmin GUI.
2. Go to *Management* > *WebAdmin Settings* > *New API Token*.
   The system generates a new API token, but you can create a custom token.
3. Map the token to an Sophos UTM user.

   Note – This cannot be the *admin* user. We recommend mapping the token to a user with reduced permissions.

4. Assign a static remote access IP to the user.
5. Under *Advanced Settings*, create a whitelist/blacklist based on IP addresses or domain names.
6. Click *Save*.

Tip – If you are using many API tokens for backend servers, you should prefix the tokens with an identifier e.g., “be0_” to easily distinguish between the keys.

3 HTTP header

The RESTful API makes use of general and custom HTTP headers to control different aspects of the interaction.

3.1 Content-Type header

When sending data via the HTTP body to the REST API, the data has to be JSON encoded.

```
content-Type: application/json
```

3.2 Accept header

The Accept header is used to encode the format that the client expects the server to produce. Currently only the JSON format is supported. The server expects the client to send the following Accept header:

```
Accept: application/json
```

3.3 Authorization header

Authorization is done using basic access authentication (see Wikipedia). The authorization header uses username and password for authentication and is supported by many services that use HTTP. You can use the username and password of any Sophos UTM user who has WebAdmin access rights. For token-based authentication (see chapter *Authenticating for API calls*), use the special username of “token”.

If you need to create the header manually with username and password authentication, use the following pseudocode as a reference:

```
“Authorization: Basic “ + EncodeBase64(username + “:” +
password)
```

If you need to create the header manually with token authentication, use the following pseudocode as a reference:

```
“Authorization: Basic “ + EncodeBase64(“token:” + token)
```

Note – Sophos UTM 9 blocks password probing. If a system tries multiple times to connect to the RESTful API with a wrong username, password, or token, Sophos UTM will block the IP address (usually for 5 minutes).

3.4 X-Restd-Err-Ack header

As described, there are multiple RESTful API interactions that can fail due to inconsistencies, e.g., object A references object B but object B is deleted. The RESTful API will prevent damage and inconsistency to confd by returning an error.

You can resolve this problem by removing the reference to object B. You can configure confd to do this automatically by setting the header value to “last”.

```
X-Restd-Err-Ack: last
```

To enable this globally for all non-fatal errors, you can set the header value to “all.”

```
X-Restd-Err-Ack: all
```

Note – Use this setting only when you’re not deleting important data or objects. Otherwise, you can acknowledge the error and cancel the operation by setting the header value to “none”. You can then troubleshoot the error with all data and objects saved.

```
X-Restd-Err-Ack: none
```

3.5 X-Restd-Lock-Override header

The confd object model supports locking objects to avoid unintended changes. To check if an object is locked or unlocked, you can use the GET method. The response will indicate the specific lock level, i.e., “_locked” can be set to “global”, “user”, or “” (empty string). A “global” value is a system lock and cannot be changed. A “user” value is a lock set by a specific user while an empty string indicates that no lock is set.

In order to change “user” lock values, you can change the lock value to an empty string and then modify the “user” value. This procedure involves three API calls and can be error prone if these calls are interrupted or not completed in the correct order. The lock override header allows the users to manually override the current value without having to change the value multiple times.

```
X-Restd-Lock-Override: yes
```

3.6 X-Restd-Insert header

In many cases when you create a new object, the object needs to be directly inserted into a node in order to be active. This usually takes two operations; however, there is an additional header you can use when creating objects to automatically activate the objects. The header will insert a reference at the given position inside the node.

For example, if you create a “packetfilter/packetfilter” rule object and need to add the object into the “packetfilter.rules” node and make it active, you can use the “1” flag to set the rule as the first rule. If you need to add the new rule object as the last rule, you can use “-1” flag. If you need the new rule object to reside somewhere in the middle, e.g., fourth rules, you can set the flag to “4”.

```
X-Restd-Insert: packetfilter.rules 4
```

3.7 X-Restd-Session header

Each interaction with the confd creates or reuses a session. Sessions are important for validation interaction and performance. However, maintaining sessions are resource intensive and can degrade performance. If you use the RESTful API to automatically create a set or predefined rules, by default Sophos UTM will maintain those sessions expecting additional API calls. You can close sessions if after creating your rules you don’t anticipate subsequent API calls for the same process. At the last step, you can set a header that will close the session.

```
X-Restd-Session: close
```

Note – `X-Restd-Session: close` may cause a longer time for the next request. Be sure to only send this command with the last request.

4 HTTP methods

Standard HTTP methods represent the different operations which you can do with the RESTful API. Each operation can report different success and error status codes.

The following list is an overview of the most common status codes for these operations:

Status code	Description
200 OK	Operation successful
201 CREATED	Operation successful and created a new resource. The newly created resource and its path and REF_ string are returned in the Location header
204 NO CONTENT	Operation successful and returned no content, e.g. when deleting a resource.
400 BAD REQUEST	The request made was invalid. The body of the response contains the error message in more detail or a resource if it is locked.
401 UNAUTHORIZED	The request is unauthorized. Add the Authorization header (see chapter *Authorization header*).
403 FORBIDDEN	The request is forbidden due to limited privileges.
404 NOT FOUND	The requested resource was not found.
422 UNPROCESSABLE ENTITY	The REST API can't handle the provided content type (see chapter *Content-Type header*).
503 SERVICE UNAVAILABLE	The REST API is not enabled (see chapter *Enabling the RESTful API*).

4.1 GET

GET requests are used to retrieve information. The GET request cannot modify the data from confd.

Examples for GET calls:

```
GET /api/nodes
GET /api/nodes/webadmin.port
GET /api/objects/network/network
GET /api/objects/network/network/REF_NetNet100008
```

4.2 PUT and POST

You can use PUT and POST for creating new resources. POST will create a new resource with an auto generated REF_ string whereas PUT will create resource for the REF_ string. You can use PUT to update the same resource after creation. PUT and POST require that you set all mandatory attributes of an object or node. You may need to override changes by removing locks (see chapter *X-Restd-Lock-Override header*).

```
PUT /api/nodes/packetfilter.rules
```

```
POST /api/objects/packetfilter/packetfilter
```

```
PUT /api/objects/packetfilter/packetfilter/REF_
PacPacAllowAnyFTPOut
```

4.3 Patch

You can use PATCH requests to update fields on an existing object:

```
PATCH /api/objects/packetfilter/packetfilter/REF_
PacPacAllowAnyFTPOut
```

4.4 Delete

You can use DELETE requests to destroy object resources that were created with POST or PUT requests.

Confd may deny DELETE requests due to validation checks. To use confd auto-correction, use the special headers described in chapter *X-Restd-Err-Ack header*.

```
DELETE /api/objects/packetfilter/packetfilter/REF_PacPacAllowAnyFTPOut
```

5 Explore API

In order to create valid API call, you can use several tools within Sophos UTM to help structure your API calls. The tools are Swagger API Documents, Swagger UI, confd client, and config-watch.plx.

5.1 Swagger API documents

You can use the Swagger API Documents to identify all the different RESTful API endpoints with descriptions for each object and node. Swagger is an open source framework that describes, produces, and visualizes RESTful web services for system. You can issue the following API call to see information for different functions for Sophos UTM:

```
GET /api/definitions
```

This returns a list of possible Swagger API definitions and you can define the call so the results are specific to different objects or nodes:

```
GET /api/definitions/network
```

The Swagger API document contains API endpoints along with parameters and object definitions for those endpoints. When objects have references to other objects the type is a regular string (REF_ string). Since not all references are allowed, each object has a description that states which subset of an object can be used as a reference. For example, the string REF(network/*) means that all network objects can be used as references while REF(network/host) means that only network host objects can be used.

5.2 Swagger UI

Swagger UI is based on the Swagger frameworks and allows you to visualize and interact with different Sophos UTM API resources. The UI provides an interactive view with drop down menus, examples, and methods to interact with the API. You can access the Swagger UI for Sophos UTM by navigating to *https://ip_address_of_UTM:4444/api/*.

Once in the Swagger UI, you can browse different nodes and objects to determine how to make an API call for Sophos UTM. For example, if you want to create a rule to block access on port 22, you would select packetfilter from the drop-down menu, enter your username and password, and select explore. Swagger UI will list all the POST operations you can use to interact with the packetfilter node.

From there, you can navigate to the different objects (in this example packetfilter/packetfilter), expand the POST and fill in parameters to block port 22. Select *Try it out!* and Swagger will display its results to be used with curl. The UI also displays Request URL, Response Body, Response Code, and Response Headers, which provide examples on how to structure the API call.

5.3 Confd client (cc)

The cc program is the command line utility used to interact with confd. cc allows navigation in the nodes tree as well as editing objects. You can use cc to understand an already existing structure or prototype some node or object configurations. To start the utility enter the cc command in an Sophos UTM shell prompt:

```
utm0:/root # cc
```

5.4 Config-watch.plx

Sophos UTM contains a tool called config-watch.plx, which monitors configuration changes on the Sophos UTM and shows which nodes where involved or which objects were created, changed, or deleted.

For example, to determine which RESTful API call to make in order to enable Web Filtering, you would perform the following steps:

1. Login to Sophos UTM via SSH.
2. Execute the file:
   ```
   utm0:/root # confd-watch.plx -v
   ```
3. Login to Sophos UTM via WebAdmin.
4. Go to *Web Protection* > *Web Filtering* > *Web Filtering Status* > *Enabled*.
5. In the SSH shell, you should see the following output:
   ```
   oc REF_DefaultHTTPProfile http profile status changed

   status = 1
   ```

Note – Created objects are displayed as `o+`, changed objects are displayed as `oc`, and deleted objects are displayed as `o-`.

6 Examples

The examples in this section use curl, a command line utility that is available for almost all operating systems to send URL-related requests.

Curl deals with HTTP and can be used in simple automation scenarios. In order to use curl in the following scenarios, the Sophos UTM certificate needs to be stored in the trusted certificates store.

6.1 Packetfilter

When creating a Packetfilter rule, you need to add the rule to the node that contains all rules in the correct order. You can use a single request including `X-RESTD-INSERT` header:

```
curl -X POST --header 'Content-Type: application/json' \
--header 'Accept: text/json' \
--header 'X-RESTD-SESSION: close' \
--header 'X-RESTD-INSERT: packetfilter.rules' \
--header 'Authorization: Basic YWRtaW46cHBwcA==' \
-d '{"action":"accept",
"destinations":["REF_NetworkAny"],
"direction":"in",
"log":true,
"services":["REF_ServiceAny"],
"sources":["REF_NetworkAny"],
"status":true}' \
'https://<UTM IP>:4444/api/objects/packetfilter/packetfilter/'
```

Note – The example uses the `X-RESTD-SESSION: close` to force closing the remote session in order to save resources.

Deleting a packet filter rule with reference `REF_PacPacXYZ`.

```
curl -X DELETE --header 'Content-Type: application/json' \
--header 'Accept: text/json' \
--header 'X-RESTD-SESSION: close' \
--header 'X-RESTD-ERR-ACK: all' \
--header 'Authorization: Basic YWRtaW46cHBwcA==' \
'https://<UTM
IP>:4444/api/objects/packetfilter/packetfilter/REF_PacPacXYZ'
```

Note – The example uses the `X-RESTD-ERR-ACK: all` to automatically approve the deletion of the object.

6.2 WebAdmin Port

Requesting the current port of the WebAdmin:

```
curl -X GET --header 'Accept: text/json' \
--header 'Authorization: Basic YWRtaW46cHBwcA==' \
'https://<UTM IP>:4444/api/nodes/webadmin.port'
```

Setting the new port

```
curl -X PUT --header 'Accept: text/json' \
--header 'Authorization: Basic YWRtaW46cHBwcA==' \
-d '6585' \
'https://<UTM IP>:4444/api/nodes/webadmin.port'
```

7 Different Sophos UTM versions

If you are using multiple Sophos UTM versions and the RESTful API, you should consider the pros and cons when choosing a versioning approach. Below are two examples of possible versioning mechanisms.

7.1 Sophos UTM versions

With this method, you query the Sophos UTM version and RESTful API to filter the available options.

Pros	Cons
Simple to implement	• Doesn't scale well with multiple versions • New capabilities in minor versions not supported

```
curl -X GET --header 'Accept: application/json' \
--header 'Authorization: Basic YWRtaW46cHBwcA==' \
'http://<UTM IP>/api/status/version'
```

7.2 Sophos UTM capabilities

You can use Swagger definitions to dynamically check the capabilities of the RESTful API.

Pros	Cons
• Codebase can support any version • More resilient to changes	More complex to start with

You can check if the RESTful API is available with a confd call definition when there is a new top level API call e.g. *status*. A value of 404 indicates that the RESTful API is not available. A value of 200 returns the RESTful API definition.

```
curl https://<UTM IP>:4444/api/definitions/status
```

If the RESTful API is available, you can use the returned document to check the availability of RESTful paths and HTTP methods. The following example uses jq in a bash script:

```
#!/bin/bash
RESULT=$(curl -s https://<UTM IP>:4444/api/definitions/status |
\
jq -r '.paths["/status/version"] | has ("get")')
if [ "$RESULT" == "true" ]
```

```
then
echo "has GET version"
else
echo "has NOT GET version"
if
```

8 References

You can review the following references for more information and examples on using the RESTful API for Sophos UTM:

- Ruby API Client Library: A lightweight Ruby library to use the RESTful API.
- Swagger UI: A resource to Swagger including documentation and definitions.
- Sophos Chef Cookbook: An integration of Sophos UTM with Chef that allows for better automatic configuration.