About the ASA REST API

First Published: December 16, 2014
Version Update Published: May 26, 2021

Contents

[hide]

Overview

This REST API provides a programmatic model-based interface for configuring classic ASAs, beginning with the 9.3.2 release. The term “classic ASAs” refers to appliances which don’t include CX or SourceFire Sensor, or the integrated functionality of NGFW (next-generation firewall) services. Also note that when other security modules are present with a classic ASA, there are no APIs for those modules.

The REST API can be used to configure ASAs in conjunction with existing management interfaces and applications–command-line interface (CLI), Adaptive Security Device Manager (ASDM) and Cisco Security Manager (CSM).

Note: Following the release of ASA REST API version 1.3.2-346, the API version numbering scheme was changed to match Cisco ASDM versioning.

New features in REST API 7.16(x):

- No new features were added. This release is only a renumber release to accompany ASA 9.16.

New features in REST API 7.15(x):

- No new features were added. This release is only a renumber release to accompany ASA 9.15.

New features in REST API 7.14(x) – (was version 1.3.2):

- This release contains bug fixes only; no new features were added.

New features in REST API 1.3.1:

- Support for additional application inspection protocols (ESMTP, SNMP).
- Certificate management.
- TLS proxy.

New features in REST API 1.2.2:

- Smart Licensing.
- Support for IP Audit and for additional application inspection protocols (FTP, NetBIOS, RTSP, SIP, SQL*Net).
- Ability to query for the ASA’s serial number.
- REST API 1.2.2.200 release includes a fix for CSCux92088: Increase the limit of bulk api request entries to 1000.

New features in REST API 1.2.1:

- Monitoring support for multi-context mode.
- Support for the following ASA features: DHCP Server and Relay, DNS Client and Dynamic DNS, Protocol Timeouts (PTO), and GTP inspection.

New features in REST API 1.1.1:

- Support Token Based Authentication.
- Support for the following ASA features: Application Inspection protocols (DNS over UDP, HTTP, ICMP, ICMP ERROR, RTSP, DCERPC, IP Options), Backup and Restore, Connection Limits, Multi-context (limited support), NTP and Write Memory command API.

Features in REST API 1.0.1:

- Support for the following ASA features: AAA, Access Rules, Failover, Interfaces, Licensing (Permanent and Activation Key Licenses), Shared Secret License, Logging, Management Access, Monitoring, NAT (Twice NAT and Object NAT), Objects, Static Routing, Service Policy and Site-to-Site VPN.
- A Bulk API.
- A Generic CLI Command Executor API, meaning any CLI commands can be sent using the REST API.

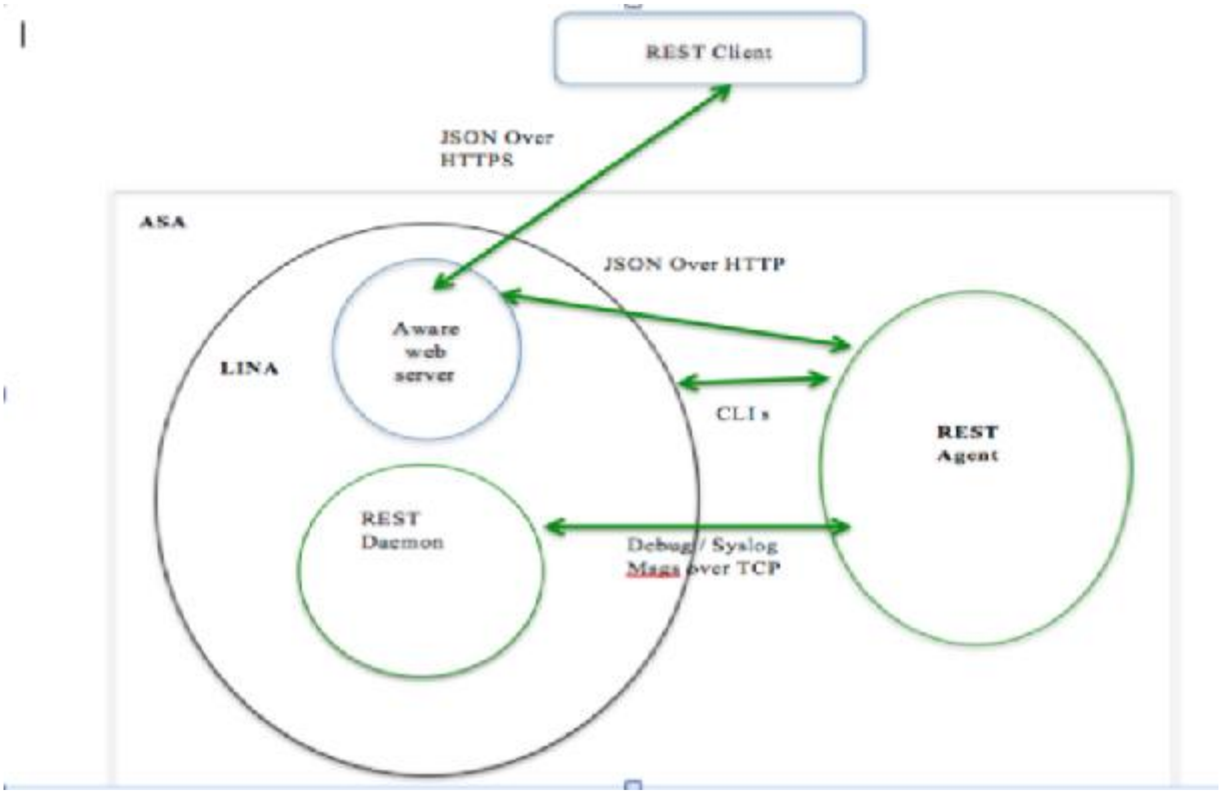
Supported Platforms

Refer to the ASA REST API Compatibility matrix for up-to-date information about ASA REST API and ASA compatibility.

Supported Modes

The REST API currently does not support direct configuration of any options in multi-mode. Only Generic CLI Command Executer API (CLI pass-through), Token Authentication API and monitoring are supported in multi-context mode. See the section “Multi-context mode” for more information.

High-level Architecture



Typical Request Flow

This is the flow for any REST PUT/POST/DELETE API request:

- REST Client establishes SSL connection to ASA.
- REST Client sends API request with basic authentication header to ASA.
- ASA HTTP server validates and processes client's request.
- ASA HTTP server opens a connection to the REST Agent using TCP channel, and writes the HTTP request to the REST Agent.
- ASA HTTP server waits for a response from the REST Agent process.
- REST Agent processes API request, picks the session/user information and invokes CLI commands request to LINA listening on 'localhost' port on ASA. REST Agent includes the session/user info in the request.
- LINA admin handler processes the CLI commands and collects resulting output.
- LINA sends the response for the CLI commands request to REST Agent.
- REST Agent prepares the response for REST API request and sends it to the ASA HTTP server.
- ASA HTTP server forwards the response to the client. The server doesn't do any processing on the response received from REST Agent process.

Resource Identity

All Resources will have a unique identifier “objectId” which will be either a natural unique name for the given type, assigned by the user, or a hash generated from composite unique attributes. Note that since the CLI has no notion of unique identifiers (UIDs), and since the REST Agent is stateless, it is not possible for the REST Agent to generate distinct unique identifiers.

Example:

```
{
  "kind": "object#AccessGroup",
  "selfLink": "https://<asa_ip>/api/access/in/inside",
  "ACLName": "inside_in_acl",
 " direction": "IN",
  "interface": {
    "kind": "objectRef#Interface",
    "refLink": "https://<asa_ip>/api/interfaces/physical/GigabitEthernet0_API_SLASH_1",
    "objectId": "GigabitEthernet0_API_SLASH_1",
    "name": "inside"
  }
}
```

Resource URL: ‘selfLink’ Attribute

The “selfLink” attribute is the complete URL for a resource, specified within the JSON definition of that object. This allows direct access to a collection of object elements, without needing to construct the URL from its objectID. This attribute will be specified in the JSON definition of every resource object.

The objectId part of the selfLink will be URL-encoded, regardless of whether the selfLink is part of a JSON response or a location header.

Upon receipt of an API request, a canonicalization check for double or mixed encodings is performed on the request URL. If the URL is double encoded, a “400 bad request” will be returned. If the URL passes the canonicalization check, the request URL is decoded and sent for further processing.

Note: The objectId within the JSON response is never URL-encoded. So, if a URL is being explicitly constructed using the objectId from a JSON response (as opposed to using selfLink), then the URL should be constructed after appropriately URL-encoding the objectId.

Resource Type: ‘kind’ Attribute

All JSON objects have a “kind” attribute indicating the type of object content–if the object represents a list, it will have a kind attribute of ‘collection#{type}’; otherwise it will be some form of ‘object#{type}’ or a primitive kind, as described in the next section.

Examples:

kind: collection#accessPolicySet => represents a list of ACL entries
kind: object#networkobject => represents an object of type ‘networkobject’

kind: objectref#networkobject => represents a reference to an object of type ‘networkobject’
kind: IPAddress => represents a primitive resource of type ‘ipAddress’

Primitive kinds

Some primitives like IP Address, Network, FQDN, Service Type, etc. can be represented using “kind” when they are mixed with other resource types. In those cases, “kind” will be without a ‘#’ and will be specified directly. Such resources will be very simple and in addition to “kind” they will include only a “value” attribute which specifies the value.

Example:

```
{
"kind: "IPv4Address "
"value": "1.1.1.1"
}
```

Resource Association

Other resources can be referenced from a given resource. There are two types of references:

1. Through an in-line object where the complete referring object is present in its entirety. This approach is used rarely and supported only in certain APIs.

2. The most common way to refer to another resource is through its resource identifier, which could be objectId or refLink.

Example:

```
{
"kind": "objectref#networkObjectGroup" ,
"refLink": "http://host/api/object/networkObjectGroups/548292" ,
"objectId":"548292"
}
OR
{
"kind": "objectref#networkObjectGroup" ,
"refLink": "http://host/api/object/networkObjectGroup/Lab%20Printers" ,
"objectId": "Lab Printers"
}
```

Object ‘rangeInfo’

Most collection resources will include a “rangeInfo” object, which provides details on the range of items contained in the collection. The GET request and Query API support pagination and will never return more than a defined maximum number of items. So if you have 20,000 network objects, you cannot get all of them in a single call. Also, in the API request you can specify the offset and the limit from that offset that should be returned in the result. This result will always contain a “rangeInfo” entry, specifying what the offset was and the limit that being returned, and the total number of items.

```
"rangeInfo": {
"offset": "integer",
"limit": "integer",
"total": "integer",
},
```

Maximum accepted value of limit will be 100. If the REST Client queries for more than 100 items, and more than 100 items are available, only 100 items will be returned, and the “total” will indicate the available-item count.

REST API Authentication

There are two ways to authenticate: Basic HTTP authentication, which passes a user name and password in every request, or Token-based authentication with secure HTTPS transport, which passes a previously created token with each request. Either way, authentication will be performed for every request. See Token_Authentication_API for additional information about Token-based authentication.

Note: Use of Certificate Authority (CA)-issued certificates is recommended on ASA, so REST API clients can validate the ASA server certificates when establishing SSL connections.

Privilege 3 or greater is needed to invoke monitoring APIs.

Privilege 5 or greater is needed for invoking GET APIs.

Privilege 15 is needed for invoking PUT/POST/DELETE operations.

REST API Conventions

- An HTTP PUT request is used to replace, update, or modify an existing resource, while HTTP POST is used to create a new resource (or any action that is not covered by PUT). You must not use HTTP PUT to create a resource.

  Note: Some types of object–for example, management access host and any ACE–are identified by a hash value which is calculated based on several of the object’s parameters. If you use HTTP PUT to change any of these parameters, the object’s hash value changes. Since this value identifies the object, it might seem that the HTTP PUT call created a new object, but this is in fact not the case.

- The request body of an HTTP PUT request must contain the complete representation of the mandatory attributes of the resource.

- An HTTP PUT accepts a complete resource. It does not return the updated version in the response. If a modified resource in not sent in the response, the HTTP status code is 204 (not 200 OK) in the HTTP header response.

- HTTP PATCH is supported where applicable to partially update a resource. Any attribute not specified will take the value of the server value.

Note: As noted for HTTP PUT, use of HTTP PATCH can change an object's identifying hash value, and as with HTTP PUT, this does not mean that the HTTP PATCH call created a new object.

- An HTTP POST request contains the details of a new resource to be created in JSON format.
- An HTTP POST response to a Create request will have a 201 return code and a Location header containing the URI of the newly created resource in the HTTP header.
- An auto-created configuration (resource) will not support a create-and-delete REST operation; that is, no HTTP POST and DELETE request. For example, you cannot create or delete the logging-related configuration, but it can be modified (PUT) or retrieved (GET).
- Neither HTTP GET nor HTTP DELETE has a request body.
- An HTTP DELETE of a collection of resources is not supported since you would be deleting the resource identified by that URL. If that resource was deleted, you would not be able to create a sub-resource (the "item" in the collection).
- An HTTP GET response has a "kind" attribute to indicate the name of the object or collection of objects.
- All REST API requests and responses must be in JSON format.
- All JSON attributes must employ the "CamelCase" naming convention; for example, "policyType."
- JSON values of type String must be in double quotes; values of type Boolean or Number need not be double quoted. A Boolean value is either true or false, in lower case.
- Every received HTTP request is expected to have this "Accept: application/json" statement in its HTTP header, indicating the REST client expects the REST response to be in JSON format.
- Every HTTP POST request must include a JSON body (an attribute).
- The Location header in the HTTP response will contain the complete URL for all the POST (create) scenarios.
- Brackets, as in [<items>] in the JSON representation of a schema, indicate a list of items.
- Unless otherwise specified, an HTTP GET returns the currently configured state.
- Whether an attribute will be shown if it has no value depends if it is an optional attribute or not. If it is optional, it can be omitted in the HTTP GET response. If it is not optional, its value will be presented as an empty string if the attribute is of type String, or as a 0 (zero) if it is a Number.
- Pagination is supported and will be restrict the maximum number of items that can be retrieved through a GET or Query API call.

REST API Codes

HTTP error codes will be reported based on these standards:

HTTP Error Code in HTTP header	Description

400 Bad Request	Invalid query parameters: unrecognized parameters, missing parameters, or invalid values.
404 Not Found	The URL does not match an existing resource. For example, an HTTP DELETE of a resource fails because the resource is unavailable.
405 Method not Allowed	An HTTP verb that is not allowed, such as a POST, on a read-only resource.
500 Internal Server Error	Server Error. A catch-all for any other failure – this should be the last choice when no other response code makes sense.

In addition to the error code, the returned response may contain a body, which includes an error object providing more details about the error.

HTTP success codes will be reported based on these standards:

HTTP Success Code in HTTP header	Description
200 Success OK	The resource was retrieved successfully using GET method.
201 Created	The resource was created successfully using POST method.
204 No Content	The resource was updated successfully using PUT or PATCH method, or deleted successfully (DELETE).

JSON Error/Warning Response Schema

```
{
  "level"   : "string",
  "code"    : "string",
  "context": "string",
  "details": "string"
}
```

Property	Type	Description
level	String	"Error," "Warning" or "Info."
code	String	Response code, such as "READ-ONLY-FIELD", or a code specific to a particular feature.

context	String	The name of the attribute to which this Error/Warning/Info response applies.
details	String	Detailed message for this Error/Warning/Info response.

REST API Agent in ASA

Installing and Enabling the ASA REST API Agent

The REST API Agent is published individually with other ASA images on cisco.com. For physical ASAs, the REST API package must be downloaded to the device’s flash and installed using the “rest-api image” command. The REST API Agent is then enabled using the “rest-api agent” command.

With a virtual ASA (ASAv), the REST API image must be downloaded to the “boot:” partition. You must then issue the “rest-api image” command, followed by the “rest-api agent” command, to access and enable the REST API Agent.

You can download the appropriate REST API package for your ASA or ASAv from https://software.cisco.com/download/home. Locate the specific *Adaptive Security Appliances (ASA)* model and then choose *Adaptive Security Appliance REST API Plugin*.

Note: The REST API Agent is a Java-based application. The Java Runtime Environment (JRE) is bundled in the REST API Agent package.

Usage Guidelines

Important: You must include the header `User-Agent: REST API Agent` in all API calls and existing scripts. Use `-H 'User-Agent: REST API Agent'` for the CURL command.

In multi-context mode, the REST API Agent commands are available only in the System context.

Maximum Supported Configuration Size

The ASA Rest API is an “on-board” application running inside the physical ASA, and as such has a limitation on the memory allocated to it. Maximum supported running configuration size has increased over the release cycle to approximately 2 MB on recent platforms such as the 5555 and 5585.

The ASA Rest API also has memory constraints on the virtual ASA platforms. Total memory on the ASAv5 can be 1.5 GB, while on the ASAv10 it is 2 GB. The Rest API limits are 450 KB and 500 KB for the ASAv5 and ASAv10, respectively.

Therefore, be aware that large running configurations can produce exceptions in various memory-intensive situations such as a large number of concurrent requests, or large request volumes. In these situations, Rest API GET/PUT/POST calls may begin failing with 500 - Internal Server Error messages, and the Rest API Agent will restart automatically each time.

The workarounds to this situation are either move to higher-memory ASA/FPR or ASAV platforms, or reduce the size of the running configuration.

‘rest-api image’ Command

This command will perform compatibility/validation checks and inform you if there are problems. If all checks pass, it will install the REST API image. To uninstall, use the "no" form of the command.

```
[no] rest-api image disk0:/<package>
```

image - Use this keyword to install/uninstall the REST API image on an ASA; provide the destination (in this case, "disk0:" for the ASA’s flash memory) and the name of the REST API image package.

Installing/updating the rest-api package will not trigger a reboot of the ASA.

This configuration will be saved in the startup config file.

Example

This example downloads the REST API package from a TFTP server and then installs it:

copy tftp://<*tftpserver*>/asa-restapi-121-lfbff-k8.SPA disk0:
rest-api image disk0:/asa-restapi-121-lfbff-k8.SPA

Supported Modes

single/multiple context, routed/transparent

Additional Boot-strapping Required for the REST API Agent

- Enable HTTP server and let clients connect over management interface:
  http server enable
  http 0.0.0.0 0.0.0.0 <*mgmt interface nameif*>

- Configure a static route on the ASA for API traffic; for example:
  route <*management interface nameif*> 0.0.0.0 0.0.0.0 <*gwip*> 1

- If command authorization is configured to use an external AAA server (for example, aaa authorization command <*TACACS+_server*>), then a user named “enable_1” must exist on that server with full command privileges.

  If command authorization is configured to use the ASA’s LOCAL database (aaa authorization command LOCAL), then all REST API users must be registered in the LOCAL database with privilege levels that are appropriate for their roles:

  - Privilege level 3 or greater is required to invoke monitoring requests.
  - Privilege level 5 or greater is required for invoking GET requests.
  - Privilege level 15 is necessary for invoking PUT/POST/DELETE operations.

‘rest-api agent’ Command

To enable the REST API Agent after installing a REST API image, use the "rest-api agent" command. To disable the REST API Agent, use the "no" form of the command.

```
[no] rest-api agent
```

agent - Starts the REST API Agent process on the ASA.

Prerequisite: HTTP server must be enabled in order for the REST API Agent to work.

When the REST API Agent is enabled, ‘/api’ URL requests are redirected from the ASA HTTP server to the REST API Agent.

Supported Modes

single/multiple context, routed/transparent

‘show rest-api agent’ Command

The "show rest-api agent" command shows the current status of the REST API Agent:

```
ciscoasa(config)# show rest-api agent
The REST API Agent is currently enabled
```

or

```
ciscoasa(config)# show rest-api agent
The REST API Agent is currently disabled
```

When the REST API Agent is disabled, ‘/api’ URL requests are rejected by the ASA HTTP server with a 404 status code response.

Supported Modes

single/multiple context, routed/transparent

‘show version’ Command

The version of the REST API Agent is listed in the output of the "show version" command:

```
ciscoasa(config)# show version
Cisco Adaptive Security Appliance Software Version 9.4(1)
REST API Agent Version <version number>
```

REST API Agent Debugging

The "debug rest-api " command enables REST API Agent debug traces on the CLI terminal.

When invoked, the command triggers a message from the REST API daemon to the REST API Agent to enable and forward debug logs. Subsequently, the REST API Agent forwards debug logs over TCP to the REST API daemon, and these logs are displayed during the CLI session. When the CLI session closes, or when the ‘no debug rest-api’ command is issued, the REST API daemon informs the REST API Agent to disable logging for the session.

```
debug rest-api [agent | cli | client | daemon | process | token-auth] {event, error}
```

agent - Debugging information for REST API Agent operations.

cli - Debugging information for REST API CLI daemon to REST API Agent communications.

client - Debugging information for Message routing between a REST API client and the REST API Agent.

daemon - Debugging information for REST API daemon to the REST API Agent communications.

process - Debugging information for REST API Agent start/stop processing.

token-auth - Debugging information for REST API Token Authentication processing.

Supported Modes

single/multi-context, routed/transparent

Output of Show Commands

"debug rest-api agent is enabled" or "debug rest-api agent is disabled"

"debug rest-api cli is enabled" or "debug rest-api cli is disabled"

"debug rest-api daemon is enabled" or "debug rest-api daemon is disabled"

"debug rest-api http is enabled" or "debug rest-api http is disabled"

"debug rest-api process is enabled" or "debug rest-api process is disabled"

"debug rest-api token-auth is enabled" or "debug rest-api token-auth is disabled"

Syslogs

Syslog #342001

Description/Rationale/Overview:

The REST API Agent was successfully started.

Default Level:

7

Syslog Number and Format:

%ASA-7-342001: REST API Agent started successfully.

Explanation:

The REST API Agent must be successfully started before a REST API Client can configure ASA.

Recommendation/Action:

None

Syslog #342002

Description/Rationale/Overview:

The REST API Agent failed.

Default Level:

3

Syslog Number and Format:

%ASA-3-342002: REST API Agent failed, reason: <*reason*>

<*reason*> The reason why the REST API Agent failed.

Explanation:

The REST API Agent could fail to start or crash for many different reasons. One reason could be that the REST API Agent is running out of memory. Another reason could be that the messaging carried out to enable/disable the REST API Agent is failing.

Recommendation/Action:

The administrator should attempt to disable ("no rest-api agent") and then re-enable the REST API Agent using "rest-api agent".

Syslog #342003

Description/Rationale/Overview:

Notification that the REST API Agent has failed and is being restarted.

Default Level:

3

Syslog Number and Format:

%ASA-3-342003: REST API Agent failure notification received. Agent will be restarted automatically.

Explanation:

The REST API Agent has failed and a restart of the Agent is being attempted.

Recommendation/Action:

None

Syslog #342004

Description/Rationale/Overview:

The REST API Agent could not be successfully started after multiple attempts.

Default Level:

3

Syslog Number and Format:

%ASA-3-342004: Failed to automatically restart the REST API Agent after five unsuccessful attempts. Use the 'no rest-api agent' and 'rest-api agent' commands to manually restart the Agent.

Explanation:

The REST API Agent failed to start after successive attempts.

Recommendation/Action:

Refer to syslog %ASA-3-342002 (if logged) to determine the reason behind the failure. Attempt to disable ("no rest-api agent") and then re-enable the REST API Agent again ("rest-api agent").

Syslog #342005

Description/Rationale/Overview:

The REST API image was successfully installed.

Default Level:

7

Syslog Number and Format:

%ASA-7-342005: REST API image has been installed successfully.

Explanation:

The REST API image must be successfully installed before starting the REST API Agent.

Recommendation/Action:

None

Syslog #342006

Description/Rationale/Overview:

The REST API image failed to install.

Default Level:

3

Syslog Number and Format:

%ASA-3-342006: Failed to install REST API image, reason: <reason>

<reason> The reason why the REST API Agent installation failed

Explanation:

The REST API image could fail to be installed for the following reasons:

version check failed | image verification failed | image file not found | out of space on flash | mount failed

Recommendation/Action:

The administrator should fix the failure and try to install the image again using "rest-api image <image>".

Syslog #342007

Description/Rationale/Overview:

The REST API image was successfully uninstalled.

Default Level:

7

Syslog Number and Format:

%ASA-7-342007: REST API image has been uninstalled successfully.

Explanation:

The old REST API image must be successfully uninstalled before a new one can be installed.

Recommendation/Action:

None

Syslog #342008

Description/Rationale/Overview:

The REST API image failed to uninstall.

Default Level:

3

Syslog Number and Format:

%ASA-3-342008: Failed to uninstall REST API image, reason: <reason>.

Explanation:

The REST API image could fail to be uninstalled for the following reasons:

unmount failed | rest agent is enabled

Recommendation/Action:

The administrator should disable REST Agent before trying to uninstall the REST API image.

Out-of-band Change Handling

If an out-of-band configuration change was observed when processing an REST API request, the configuration will be reloaded to the REST API Agent before attempting to process the request.

Supported ASA Features

AAA

The AAA API supports configuring the AAA-related features of authentication, authorization, and command privileges.

AAA server groups and accounting are not yet supported.

Authentication

api/aaa/authentication

Configure network authentication.

Limitations:

Currently, only the LOCAL server group is supported.

Authorization

api/aaa/authorization

Configure network authorization.

Note: If AAA authorization is configured to use an external AAA server (for example, aaa authorization command <*TACACS+_server*>), the REST Agent requires that a user named “`enable_1`” with full command privileges be defined on that AAA server. See “Enable and Configure the REST API Agent” in the *Cisco ASA REST API Quick Start Guide* for more information.

Command Privileges

api/aaa/commandprivileges

Configure the local command privilege levels.

Limitations:

N/A

Access Rules

/api/access

Use the Access API to configure network access in both routed and transparent firewall modes.

With the REST API you can GET access group access rules. The access groups are automatically created when the first access rule is created for a particular interface and direction. Similarly, an access group is deleted when its last access rule is deleted. Global access rules are supported as well.

With the REST API you can GET/POST/PUT/PATCH/DELETE access rules. The access URIs are grouped per interface and direction, and have a common URI root of /access.

Note: If you create an access control rule that references one or more service objects, those service objects must already exist. In other words, create any necessary service objects in advance of the access control rules.

Limitations:

No limitations; supports the same features as the ASDM application.

Back Up and Restore

Use this API to back up the configuration on the ASA: /api/backup

Use this API to restore the configuration on the ASA: /api/restore

Limitations:

N/A

Certificate Management

/api/certificate

Use this API to generate and manage key pairs, identity certificates and CA certificates.

Limitations:

N/A

DHCP

/api/dhcp

Use this API to configure DHCP client and DHCP relay.

Limitations:

DHCP relay is not supported in transparent mode.

DNS

/api/dns

Use this API to configure DNS.

Limitations:

N/A

Failover

/api/failover

Limitations:

N/A

Interfaces

There are six sets of APIs that can be used to provide interface-related configuration. These are for physical interfaces (/api/interfaces/physical), VLAN interfaces (/api/interfaces/vlan), port-channel interfaces (/api/interfaces/portchannel), redundant interface (/api/interfaces/redundant), bridge group interfaces (BVI) (/api/interfaces/bvi), which is available in transparent mode, and global interface set-up (/api/interfaces/setup).

Limitations:

N/A

IP Audit

/api/firewall/ipaudit

Limitations:

N/A

Licensing

Permanent and Activation Key Licenses

api/licensing/activation

APIs for viewing and configuring key-based licenses. The permanent license is retrieved via GET just as the activation licenses are.

Limitations:

The ASA must be manually reloaded after changes to the activation license configuration; for example, a new license is added, or licenses are enabled/disabled.

Shared License

api/licensing/shared

APIs to support configuring shared license settings, either client or server shared license, as defined by the active license.

Limitations:

N/A

Smart License

api/licensing/smart

API to configure smart licenses and to monitor entitlements on supported platforms.

Note that a POST request to api/licensing/smart/asav/register returns code 201 (success) even for an invalid token ID. The ASAv itself cannot validate the token ID; it relies on the License Server for validation. But calls to the License Server are issued and processed asynchronously, after the token ID is accepted by the ASAv.

Limitations:

N/A

Logging

Syslog Server

api/logging/syslogserver

API to support CRUD operations for syslog servers.

Limitations:

N/A

Syslog Server Settings

/api/logging/syslogserversettings

API to support advanced settings for syslog servers, including configuring the logging queue and permitting TCP logging when the syslog server is down.

Limitations:

N/A

Syslog Message Configuration

/api/logging/syslogconfig

API to support configuring syslog message details, including level and enabling/disabling a message.

Limitations:

N/A

Syslog Message Settings

/api/logging/syslogconfigsettings

API to support configuring syslog message settings, such as including the device ID in non-EMBLEM format, time-stamp, or cluster IP (when applicable).

Limitations:

N/A

Netflow Configuration

/api/logging/netflow

API to support CRUD operations for Netflow configuration.

Limitations:

N/A

Netflow Collector Settings

API to support CRUD operations for Netflow collector settings.

Limitations:

Service policy rules with Netflow not supported

Management Access

General management access

api/mgmtaccess

Use this API to configure ASA access settings related to telnet, SSH, and HTTPS (ASDM).

Limitations:

N/A

Hosts

/api/mgmtaccess/hosts

Allows CRUD operations on management access hosts for telnet, SSH, and HTTPS (ASDM) connections.

Limitations:

N/A

Monitoring

/api/monitoring/

These APIs can be used to get health, performance and REST API Agent monitoring statistics.

In multi-context mode, to get monitoring statistics for a given context, including the System context, append a query with a 'context' parameter: https: //<asa_admin_context_ip>/api/cli?context=<context_name>. If the 'context' query parameter is not present in a monitoring request, the REST API Agent attempts to determine the target context on its own. For resources that are available only in the System context, such as the CPU process usage, the request is directed to the System context. The rest of the commands are directed to the admin context.

Limitations:

N/A

Multi-context mode

Multi-context mode support is limited to the Generic CLI Command Executer API, Token Authentication API and monitoring. At this time, the REST API does not support configuring an ASA in multi-context mode, except via the CLI command executer API.

Notes:

- The REST API Agent can be enabled in multi-context mode. The REST API Agent CLIs are present only in the System context.
- If token authentication is used, you need to get the authentication token via https://<asa_admin_context_ip>/api/tokenservices before issuing any REST API commands.

  Note that the token received for the admin context can be used to configure/monitor any other context as well.
- Generic CLI Command Executer API can be used to configure any context as https://<asa_admin_context_ip>/api/cli?context=<context_name>. If the 'context' query parameter is not present, the request is directed to the admin context.
- If the 'context' query parameter is not present in a monitoring request, the REST API Agent attempts to determine the target context on its own. For resources that are available only in the System context, such as

the CPU process usage, the request is directed to the System context. The rest of the commands are directed to the admin context.

Limitations:

REST API commands are available only in the System context. The REST API Agent must be restarted when the ASA is switched from single- to multiple-context mode, or vice versa.

NTP

/api/devicesetup/ntp/

Limitations:

N/A

NAT

/api/nat

NAT API supports TwiceNAT (also known as Manual NAT) and ObjectNAT (also known as AutoNAT). Each NAT type has a unique URI. Before and After AutoNAT is fully supported (Routed and Transparent mode).

Attributes for configuring InterfacePAT, DynamicPAT (hide), and PAT Pool are also included in the API.

A single list showing all NAT types (Twice and Auto) in the same list is not supported.

ObjectNAT (AutoNAT)

Limitations:

Creating an in-line network object with a NAT rule is not supported. To create an object NAT for an existing network object, the source Address should point to a network object to be translated.

TwiceNAT (Manual NAT)

Before NAT and After NAT are separated into two lists and have their own URIs. Moving a Before NAT rule to an After NAT rule, or vice-versa is not supported.

Limitations:

N/A

Objects

/api/objects/

Objects are re-usable configuration components. They can be defined and used in ASA configurations in the place of in-line IP addresses, services, names, and so on. The REST API provides support for the following types of objects:

- Extended ACLs. Similar to access rules, extended ACLs are created when their first ACE is created, and are deleted when their last ACE is removed.
- Local users and user groups.

- Network objects and object groups.
- Network services (including predefined network services) and server groups. Predefined service objects cannot be changed or deleted. They can be used to cut and paste in-line services, or when creating a service object.
- Regular expressions.
- Security object groups.
- Time ranges.
- User objects.

Similarly to ASDM, the REST API supports use of in-line objects and object groups in access, NAT and service-policy rules.

Limitations:

Only local users are supported.

Protocol Timeouts

/api/firewall/timeouts

APIs to configure global protocol and session timeouts.

Limitations:

N/A

Routing

/api/routing/static

Only static routes are supported at this time.

Limitations:

N/A

Service Policy

/api/servicepolicy/

The REST API supports the following protocol inspections:

DCERPC
DNS over UDP
ESMTP
FTP
GTP
HTTP
ICMP
ICMP ERROR
IP Options

NetBIOS
RTSP
SIP
SNMP
SQL*Net

The regular expressions and connection limits are supported as separate resource URIs.

Limitations:

N/A

TLS

/api/ firewall/tlsproxy/

API to configure TLS Proxy.

Limitations:

N/A

VPN

/api/vpn/

Only Site-to-Site VPN configuration is supported in the REST API. IPv4 and IPv6 are both supported. Site-to-Site VPN monitoring is not supported.

Limitations:

Only Site-to-Site configuration is supported. Certificate Management as seen in ASDM is not supported.

Special APIs

Bulk API

As a convenience, this API lets you group multiple POST, PUT, PATCH, and DELETE requests for different resources into a single HTTP POST call. This means you can make a single request to modify multiple resources, with each contained request being processed in order of appearance in the payload. However, note that the content of a bulk request is treated as an atomic configuration change: if any of the requests within it fail, the whole payload is rejected, and no changes are made to the ASA configuration.

The details of the request payload and response structure are as follows:

POST URL: /api

Request payload format: [{}, {}, {}, ...] where each JSON object is an operation wrapper:

```
{
 method:<HTTP_REQUEST_METHOD_FOR_RESOURCE >,
 resourceUri:<RESOURCE_URI>,
 data:<POST_CONTENT_FOR_THIS_URI_IF_APPLICABLE>
}
```

Property	Type	Description
method	string	'GET', 'POST', 'DELETE', 'PATCH' calls are supported.
resourceUri	string	The resource URI if the request was made independently.
data	string	JSON data sent as raw body if the request was made individually. For the 'DELETE' method, this is not needed.

The bulk request response format is:

```
{
entryMessages:[{}, {}, ...],
commonMessages: []
}
```

entryMessages is an array of objects, with each object corresponding to a bulk request entry.

Generic CLI Command Executer API

This special API can take single- or multi-line CLI commands and present the CLI output as the API response.

POST URL: /api/cli

Request payload format:

```
{
"commands": ["command-1", "command-2",…, "command-n"]
}
```

Response format:

```
{
"response": ["command-1 response", "command-2 response",…, "command-n response"]
}
```

Limitations

The debug commands are not supported in CLI pass-through. All debug commands are per terminal session, and not a global configuration. So, if debug commands are sent over CLI pass-through, either they might return an error or success response, but they do not have any effect on the device.

Token Authentication API

The Token Services operations can be used for token-based authentication as an alternate to Basic authentication. Note that each token is generated and maintained by the ASA, and the REST API agent does not maintain any details of a token. Also, tokens must be regenerated every time the REST API agent is restarted.

The REST API client needs to send a POST request to '/api/tokenservices' with user information in the basic authentication header to get a token for that user. Subsequently, the REST API client can use this token in an 'X-Auth-Token' request header for any subsequent REST API calls. The 'token' will be valid until either it is explicitly invalidated by a 'DELETE /api/tokenservices/<token>' request, using user information in the basic authentication header, or until the session times out.

POST URL: /api/tokenservices

Request payload is empty. The user information should be in the basic authentication header.

Response could be:

Reason	HTTP Status Code
AAA validation failure/Authorization header not present.	401 Unauthorized
Authentication success.	204 No Content + X-Auth-Token <token id> (header)
Can’t get user name/password from the header, or any other sanity check failures.	400 Bad Request
Maximum sessions reached. Note: The maximum number of sessions per context is 25.	503 Service unavailable

To delete a token, issue DELETE to URL: /api/tokenservices/<token>

Request payload is empty. User information should be in basic authentication header.

Response could be:

Reason	HTTP Status Code
AAA validation failure/Invalid token.	401 Unauthorized
Success.	204 No Content
Can’t get user name/password from the header, or any	400 Bad Request.

other sanity check failures.	

Notes:

- The existing syslogs 605004 and 605005 are used for create/delete a token.
- The existing syslog 109033 is used for the case where “Challenge” is requested by the authentication server to inform the user that it is “unsupported.”
- When a REST API request is received, it is checked first for an 'X-Auth-Token' header; if it not present, the server falls back to basic authentication.
- Token authentication does not conform to the Oauth 2.0 RFC 6749 specification.
- The generated token database will be in memory on the ASA, and will not be replicated across failover pairs or clusters. In other words, if failover happens within a failover pair, or a cluster master device changes, authentication will need to be performed again.
- For a multi-context device, the token is received for the admin context and it can be used for configuring any other context as well.

Write Memory API

Changes to the ASA configuration made by REST API calls are not persisted to the start-up configuration; that is, changes are assigned only to the running configuration. This 'Write Memory API' can be used to save the current running configuration to the start-up configuration.

POST URL: /api/commands/writemem

Request payload is empty.

REST API Online Documentation

The on-line documentation interface (“Doc-UI”) combines the functionality of a user interface with all the information contained in the embedded API documentation. The Doc-UI can be run in any of the following browsers: Chrome (current), Firefox (current), Internet Explorer 9+, Safari 5.1+, Opera (current). Older versions may work, but Internet Explorer 8 and below will not.

The REST API Agent must be enabled to access the Doc-UI; the Doc-UI is accessible from `https://`*`<asa management interface ip>`*`/doc/` (note that the ending '/' is necessary for accessing the Doc-UI).

Note: When you access the local REST API documentation pages, your browser sends a request to the ASA for the pages, and also requests certain jQuery and JSON files from various Web locations. One of these locations is `https://cdnjs.cloudflare.com`.

However, when passing through an ASA with FirePOWER Services enabled, such requests may be blocked by the FirePOWER module if there is a “Categories: ad portal” blocking filter configured. To unblock the `cloudflare` site, create an access control rule that explicitly allows this site, and place it above the rule which includes an application condition blocking “ad portals.”

See http://www.cisco.com/c/en/us/support/docs/security/firesight-management-center/117956-technote-sourcefire-00.html#anc9 for information on excluding a Website/Web application from blocking due to URL filtering or application control.

Types of Scripts

Three types of scripts can be generated from the Doc-UI so you can automate REST API operations: Javascript, Python, and Perl.

Prerequisites for Using Generated Scripts

The JavaScript scripts require the installation of Node.js, which can be found at http://nodejs.org/. Node.js lets you use JavaScript applications, typically written for a browser, like a command-line script (such as Python or Perl). Simply follow the installation instructions, and then run your script with:

```
node script.js
```

The Python scripts require you to install Python, found at https://www.python.org/. Once you have installed Python, you can run your script with:

```
python script.py <username> <password>
```

The Perl scripts require some additional set-up. You will need five components: Perl itself, and four Perl libraries:

Perl, found at http://www.perl.org/

Bundle::CPAN, found at http://search.cpan.org/~andk/Bundle-CPAN-1.861/CPAN.pm

REST::Client, found at http://search.cpan.org/~mcrawfor/REST-Client-88/lib/REST/Client.pm

MIME::Base64, found at http://perldoc.perl.org/MIME/Base64.html

JSON, found at http://search.cpan.org/~makamaka/JSON-2.90/lib/JSON.pm

Here is an example Perl installation on a Macintosh:

```
Boot strapping for MAC:
$ sudo perl -MCPAN e shell
cpan> install Bundle::CPAN
cpan> install REST:: Client
cpan> install MIME::Base64
cpan> install JSON
```

After installing the dependencies, you can run your script with:

```
perl script.pl <username> <password>
```

Legal Information

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

All printed copies and duplicate soft copies are considered un-Controlled copies and the original on-line version should be referred to for latest version.

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco website at www.cisco.com/go/offices.

Cisco Trademarks

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: www.cisco.com/go/trademarks. Third-party trademarks mentioned are the property of their respective owners. The use of the word "partner" does not imply a partnership relationship between Cisco and any other company. (1110R)