About the ASA REST API v1.2.1

First Published: December 16, 2014
Revised: July 29, 2015

Contents

[hide]

Overview

REST API provides a programmatic model-based interface to configure classic ASA starting from 9.3.2 release. The term 'classic ASA' refers to the ASA which doesn't include CX or SourceFire Sensor or integrated functionality of NGFW. Also when other modules are present with classic ASA, there are no APIs for those other modules.

The REST API can be used to configure ASA together with existing management interfaces (CLI, ASDM and CSM).

Following are the new features in REST API 1.2.1 release.

- 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 support.

Following are the new features in REST API 1.1.1 release.

- 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.

Following are the features in REST API 1.0.1 release.

- Support for the following ASA features: AAA, Acces 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.
- Provide Bulk API.
- Provide a Generic CLI Command Executor API using which any CLI commands can be sent using REST API.

Supported platforms

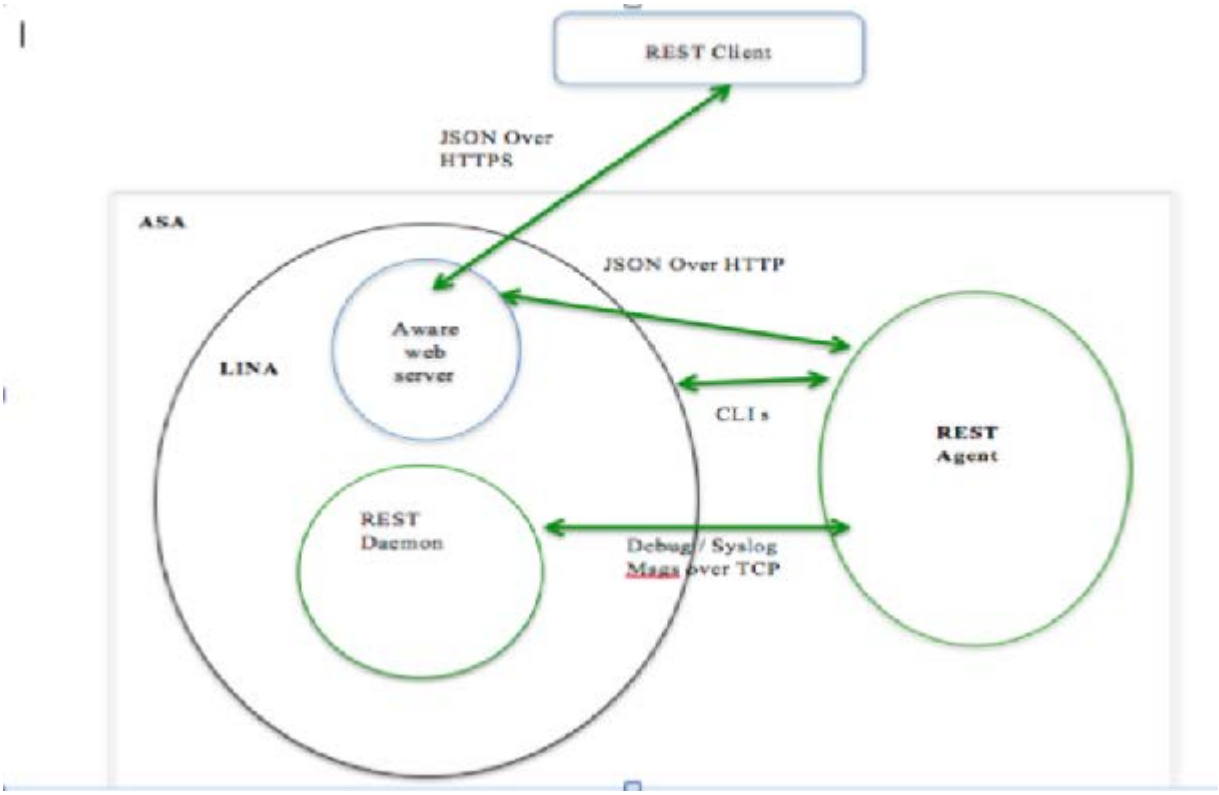
The REST API is supported only on the 5500-X series (including the 5585-X) and ASAv platforms, and Firepower 9300 ASA Security Modules; it is not supported on ASA Service Modules (ASA-SMs). See ASA REST API Compatibility for more information.

Supported Modes

The REST API currently does not support configuration of any options in multi-mode. Monitoring and CLI pass-through are supported in multi-mode.

The REST API provides limited support for multi-context mode. Only Generic CLI Command Executer API, 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

The following 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 the connection to 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 info and invokes CLI commands request to LINA listening on localhost port in ASA. REST Agent includes the session/user info in the request.
- Lina admin handler processes the CLI commands and collects the results output.
- Lina sends the response for the CLI commands request to REST Agent.
- REST Agent prepares the response for REST API request and sends to the ASA HTTP server.
- ASA HTTP server forwards the response to the client. 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 given by user or a generated hash out of composite unique attributes. Note that CLI has no notion of UID so it's not possible for REST Agent to generate any distinct unique identifier since REST Agent is stateless.

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"
  }
}
```

Attribute 'selfLink'

This is complete URL for a resource specified within the JSON object of an object. This is useful when a collection is retrieved to traverse to individual items without figuring out through documentation on how to construct URL to reach to an object from its objectId. This attribute will be specified in JSON object of every resource.

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

Whenever an API request comes, first canonicalization check will be done on the request URL to check for any double or mixed encodings. If URL is double encoded, 400 bad request will be returned. If it passes canonicalization check, then the request URL is URL-decoded and sent for further processing.

Note: The objectId within the JSON response will never be URL encoded. So, instead of using selfLink, if URL is being explicitly constructed using the objectId from JSON response, then it should be constructed after appropriately URL encoding the objectId.

Resource Type – 'kind' attribute

All objects represented in JSON will have a 'kind' attribute specifying the type of the object content. If the object represents a list then it will have syntax as 'collection#{type}' otherwise it will be 'object#{type}'

Examples:

'kind': 'collection#accessPolicySet' => represents list of ACL type.
'kind':'object#networkobject' => represents 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 could be represented using 'kind' as well when they are mixed with other resource types. In those cases the 'kind' will be without any '#' and will be specified directly. Such resources will be very simple and besides 'kind' they will only contain 'value' attribute, which specifies the value.

Example:

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

Resource association

Other resources could be referenced from a given resource. There are two type of referencing:

1. Through inline 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 contain a rangeInfo object in it, which will provide details on the range of items currently contained in the collection. The GET and Query API support pagination and will never return more than a predefined MAX number of items. So if you have 20,000 network objects, you cannot get all of it in one 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. The result will always contain a rangeInfo specifying what was the offset and limit that are being returned and the total items.

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

Maximum accepted value of limit will be 100; if REST Client queries for more than 100 items, if more than 100 items available, only 100 items will be returned and total will indicate the available item count.

REST API Authentication

HTTP Basic authentication or Token-based authentication with secure HTTPS transport are supported, and authentication will be performed for every request.

Note: It is recommended to use Certificate Authority (CA) issued certificates on ASA, so that REST API clients can validate the server certificates of ASA during the SSL connection establishment.

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; for example, 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 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 standards:

HTTP Error Code appearing in HTTP header	Description
400 Bad Request	Invalid query parameters – unrecognized parameters or, missing parameters, or invalid values.
404 Not Found	The URL does not match a resource that exists. For example, a 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.

HTTP success codes will be reported based on standards:

HTTP Success Code appearing in HTTP header	Description
200 Success OK	The resource is 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).

In addition to the error code, the return response may contain body, which will have error object containing more details about the error as appropriate.

JSON Error/Warning Response Schema

```
[
  { “code”   : “string”,
    “details”: “string”,
    “context”: attribute name,
    “level”  : <Error/Warning/Info>
  },
  ...
]
```

Property	Type	Description
messages	List of Dictionaries	List of error or warning messages
code	String	Error/Warning/Info code
details	String	Detailed message corresponding to Error/Warning/Info

REST Agent in ASA

Installing and Enabling REST Agent in ASA

With the current rest-api implementation, JRE and rest-api agent are bundled in the ASA image. This causes significant increase in the size of ASA image from 9.3.1 release. And with this approach, Java is bundled by default in ASA images and this caused concerns from field/sales engineers.

To address this issue, through rounds of discussions, it is decided that we will package rest-api + JRE together, sign and publish separately on cisco.com. (New keys will be needed signing the REST package) This way, shipped ASA images will not have rest-api plugin package. Only the customers needing rest-api will download the separate package, put it on flash and invoke the CLI command to start rest-api agent. As a result, we plan to add the following CLI to install/uninstall rest-api image. Users still need to invoke the existing "[no] rest-api agent" command to enable/disable rest-api agent after the installation. But per customer requirement, "[no] rest-api agent" will now be part of the running config instead of storing in a file on flash.

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

This command will perform compatibility/validation checks and inform if there are problems. If all checks pass, it will install the rest-api image.

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

This config will be saved in the startup config file.

"clear configure" will clear this config.

To uninstall, use the "no" form of the command.

Syntax & Help message

image - Use this keyword to install the REST API image on ASA.

Output of Show Commands

REST API image file, <path to the file>
E.g. CLI sequence for installing rest-api image:
copy tftp://<tftpserver/asa-restapi-9.3.2-32.pkg disk0:
rest-api image disk0:/asa-restapi-9.3.2-32.pkg
E.g. CLI sequence for uninstalling rest-api image:
no rest-api image disk0:/asa-restapi-9.3.2-32.pkg
E.g. CLI sequence for Upgrading rest-api image:
no rest-api image disk0:/asa-restapi-9.3.2-32.pkg
copy tftp://<tftpserver/asa-restapi-9.3.2-33.pkg disk0:
rest-api image disk0:/asa-restapi-9.3.2-33.pkg

REST Agent is a process based on ASDM code. By default REST Agent process will not be started in ASA. A new CLI command when invoked by user will start the REST Agent process in ASA.

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

rest-api agent

Will start the REST Agent process in ASA. Prerequisite: HTTP server should be enabled prior to this. If HTTP server was not enabled warning will be printed in the CLI.

When enabled '/api' URL requests will be redirected from ASA HTTP server to the REST Agent.

no rest-api-agent

Will stop the REST Agent process in ASA.

When disabled, '/api' URL requests will be rejected by ASA http server with 404 status code response.

Additional boot-strapping required for REST 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>'
- Set the authentication approach for HTTP: 'aaa authentication http console LOCAL'
- Create a local user with privilege 15 (for read/write operations): 'username <user> password <pass> encrypted privilege 15
- Configure (static) routes

REST Agent debugging

" debug rest-api agent {event | error}" CLI command will enable and show the REST API Agent debug traces on CLI.

When invoked the above command will trigger message from REST Daemon to REST Agent for enabling and forwarding the debug logs. Subsequently REST API Agent will forward debug logs over TCP to REST API Daemon; and these logs will be displayed on CLI session. When the CLI session closes or when 'no debug rest-api agent' CLI command invoked, REST daemon will inform REST Agent to disable logging for the session.

CLI commands

Debugging REST API modules / agent

CLI:

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

Syntax & Help message:

rest-api	REST API information
- agent	REST API Agent debugging information
- cli	REST API CLI Daemon to REST API Agent communication debugginginformation

- *client* Message routing between a REST API Client and the REST API Agent debugging information
- *daemon* REST API Daemon to REST API Agent communication debugging information.
- *process* REST API Agent process start/stop debugging information
- *token-auth* *REST API Token Authentication debugging information*

Supported Modes:

single/multi-context, routed/transparent

Output of Show Commands:

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

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

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

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

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

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

Enabling or Disabling REST API Agent

CLI:

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

Description:

Use this command to enable the REST API Agent residing on ASA. This config command is slightly different from other config commands:

- This config is saved in a separate file on flash, not in the startup config file.
- Once enabled, " clear configure" will not clear this config.
- To disable, use the " no" form of the command.

Syntax & Help message:

agent Use this keyword to enable the REST API Agent on ASA.

Supported Modes:

single/multi-context, routed/transparent

Output of Show Commands

" The REST API Agent is currently enabled" / " The REST API Agent is currently disabled"

Another way to find out if the REST API Agent is enabled is by issuing the " show version" command.

Version info for REST agent

The " show version" output is modified when REST API Agent is enabled.

CLI:

```
Saleen6(config)# show version
Cisco Adaptive Security Appliance Software Version 100.10(0)84
REST API Agent Version <version number>
```

Syslogs

REST API Install Syslogs

Syslog #1

Description/Rationale/Overview:

The REST API image has been successfully installed.

Default Level:

7

Syslog Number and Format:

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

Rate Limited:

No

Explanation:

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

Recommendation/Action:

None

Syslog #2

Description/Rationale/Overview:

Failure reason behind the REST API image installation is reported.

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

Rate Limited:

No

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 #3

Description/Rationale/Overview:

The REST API image is successfully uninstalled.

Default Level:

7

Syslog Number and Format:

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

Rate Limited:

No

Explanation:

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

Recommendation/Action:

None

Syslog #4

Description/Rationale/Overview:

The REST API image fails to uninstall.

Default Level:

3

Syslog Number and Format:

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

Rate Limited:

No

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.

REST API Agent Syslogs

Syslog #1

Description/Rationale/Overview:

The REST API Agent was successfully started.

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 #2

Description/Rationale/Overview:

The reason behind the REST API Agent failure is reported.

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 enable the REST API Agent again using " rest-api agent" .

Syslog #3

Description/Rationale/Overview:

A notification that the REST API Agent has failed.

Syslog Number and Format:

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

Explanation:

A notification of the REST API Agent is received and a restart of the Agent is being attempted.

Recommendation/Action:

None

Syslog #4

Description/Rationale/Overview:

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

Syslog Number and Format:

%ASA-3-342004: Failed to automatically restart the REST API Agent after 5 unsuccessful attempts. Use the 'no rest-api agent' and 'rest-api agent' commands to manually restart the Agent.

Explanation:

The REST API Agent has failed to start after many attempts.

Recommendation/Action:

Administrator should refer should to syslog %ASA-3-342002 (if logged) to better understand the reason behind the failure. The administrator should attempt to disable (" no rest-api agent") and then enable the REST API Agent again using " rest-api agent" .

Out-of-band changes handling

When processing the REST API request if out-of-band configuration change was observed, configuration will be reloaded in to REST API Agent before further handling the request.

Typical request flow

The following 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 authenticates client's request.
- ASA HTTP server opens the connection to REST Agent using TCP channel, and writes the HTTP request to the REST Agent.
- ASA HTTP server waits for REST Agent process's response.
- REST Agent processes API request, picks the session/user info and invokes CLI commands request to LINA listening on localhost port in ASA. REST Agent includes the session/user info in the request.
- Lina admin handler processes the CLI commands and collects the results output.
- Lina sends the response for the CLI commands request to REST Agent.
- REST Agent prepares the response for REST API request and sends to the ASA http server.
- ASA HTTP server forwards the response to the client. Server doesn't do any processing on the response received from REST Agent process.

Supported ASA Features

AAA

The AAA API currently supports configuring AAA-related features of authentication, authorization, and command privilege.

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.

Limitations:

Currently, only the LOCAL server group is supported.

Command Privileges

api/aaa/commandprivileges

Configure the local command privilege levels.

Limitations:

N/A

Access Rules

/api/access

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

With REST API you can GET access groups 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 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:

Limitations:

No limitations; support same features as the ASDM GUI.

Backup and Restore

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

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

Limitations:

N/A

DHCP

Use these APIs to configure DHCP client and DHCP relay: /api/dhcp

Limitations:

DHCP relay is not supported in transparent mode.

DNS

Use these APIs to configure DNS: /api/dns

Limitations:

N/A

Failover

/api/failover

Limitations:

N/A

Interfaces

There are six set of URLs that can be used to make interface related configuration. They are for physical interface, vlan interface, port-channel interface, redundant interface, bridge group interface (bvi) which is available in transparent mode, and global interface setup, respectively.

Limitations:

N/A

Licensing

Permanent and Activation Key Licenses

api/licensing/activation

API to support viewing and configuring key-based licenses. The permanent license is retrieved via GET just like the activation licenses.

Limitations:

ASA must be manually reloaded after changes to the activation license configuration (e.g., new license added, licenses enabled/disabled).

Shared License

api/licensing/shared

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

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 permiting 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, timestamp, 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 REST APIs can be used to get health, performance and REST 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 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:

For cluster member and resource usage info, queries should be made to the cluster master only.

Multi-context mode

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

Notes:

- The REST agent can be enabled in multi-context mode. The REST 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 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 agent must be restarted when 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 (aka Manual NAT) and ObjectNAT (aka AutoNAT). Each NAT type has its unique URI. Before and After autoNAT is fully supported (Routed and Transparent mode).

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

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

ObjectNAT (AutoNAT)

Limitations:

Creating an inline network object with NAT rule is not supported. To create an object NAT for an existing network object, source Address should point to 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 to an After NAT or vice-versa is not supported.

Limitations:

N/A

Objects

/api/objects/

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

- Extended ACLs. Similarly to access rules, extended ACLs are created, when their first ACE is created, and are deleted, when their last ACE is deleted.
- Local users and user groups
- Network objects and object groups
- Network services (including predefined network services) and server groups. Predefined service object cannot be changed or deleted. They can be used to cut and paste as inline services or when creating a service object.
- Regular expressions
- Security object groups
- Time ranges
- User objects

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

Limitations:

Only local users are supported.

Protocol Timeouts

/api/firewall/timeouts

Use these 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
HTTP
ICMP
ICMP ERROR
IP Options
RTSP

The regular expression, connection limits are supported as separate resource URIs.

Limitations:

N/A

VPN

/api/vpn/

Only S2S VPN configuration is currently supported in Rest API. IPv4 and IPv6 are both supported. S2S VPN Monitoring is not supported.

Limitations:

Only S2S Configuration is supported. Certificate Management as seen in ASDM is not yet supported.

Special APIs

Bulk API

This API allows to group multiple POST, PUT, PATCH, DELETE for different resources in a single HTTP POST call. This is for optimization purposes to make a single request to modify multiple configurations. This will provide a wrapper where individual request for each resource data could be grouped together in a single JSON content. It might provide atomicity for certain level of validations where the complete request is rejected if syntax validations fail for some.

This digresses from the REST paradigm although each individual operation specified will follow REST convention and users don't need to relearn new API but just how to package up multiple REST calls in bulk.

The details of request payload, response structure are as follows.

Post URL: /api

Post request payload: [{}, {}, {} ..] – each JSON object is an operation wrapper. The object wrapper is given below.

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

Property	Type	Description
method	string	Supported are 'GET', 'POST', 'DELETE', 'PATCH'
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 'DELETE' method, this is not needed.

The bulk request response structure will be as follows.

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

Entry messages will be an array of objects, with each corresponding to the bulk entry.

Generic CLI Command Executer API

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

Post URL:

/api/cli

Post Request Payload/Content:

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

Response:

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

The details of request payload, response structure are as follows. POST URL: /api/tokenservices

Post request payload is empty. The user information should be in basic authentication header. The response could be as follows.

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 username/password from the header or any other sanity check failures	400 Bad Request.
Maximum sessions reached	503 Service unavailable

Note: The maximum sessions per context is 25.

To delete a token. DELETE URL: /pai/tokenservices/<token>

Request payload is empty and user information should be in basic authentication header. The response could be as follows.

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 other sanity check failures	400 Bad Request.

Notes:

The existing syslogs 605004 and 605005 will be used for create/delete a token.

Existing syslog 109033 will also be 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, first it checks for 'X-Auth-Token' header and if it not present then it will fall back to basic authentication.

The token authentication will not conform to Oauth 2.0 RFC 6749 specification.

The generated token database will be in the memory on ASA and will not be replicated across failover pair or clustering. What this means is that, if in within a failover pair, failover happens or cluster master device changes, the authentication needs to be performed again.

For a multi-context device the token is received for 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 startup configuration; that is, changes are assigned only to the running configuration. This 'Wirte Memory API' can be used to save the current running configuration to the startup 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).

Types of Scripts

There are three types of scripts that can be generated by the Doc-UI that will allow you to perform 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 allows you to use Javascript, typically written for a browser, like a command line script (such as Python or Perl). Simply follow the installation instructions, and 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 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)