SimpleAciUiLogServer Documentation

Release latest

August 16, 2015

Contents

A Simple HTTP/HTTPS server that accepts POSTs from the APIC UI as a remote API Inspector.

The simplest method to use this module is to execute it as a standalone script, however it is designed to allow it to be imported as a module as well.

<http://datacenter.github.io/SimpleAciUiLogServer>

CHAPTER 1

Standalone Script

```
$ SimpleAci
SimpleAciUiLogServer     SimpleAciUiLogServer.py
$ SimpleAciUiLogServer
serving at:
http://10.10.10.107:8987/apiinspector
https://10.10.10.107:8443/apiinspector

2015-01-25 05:07:11,040 DEBUG -

    method: GET
       url: http://172.1.1.176/api/node/class/fabricTopology.json?subscription=yes
   payload: None
    # objs: 1
  response: {"totalCount":"1","subscriptionId":"72057761559216131","imdata":
  [{"fabricTopology":{"attributes":{"childAction":"","dn":"topology",
  "lcOwn":"local","modTs":"2015-01-08T02:10:36.147+04:00","monPolDn":
  "uni/fabric/monfab-default","status":""}}}]}
```

The standalone script can be invoked using any of these commands:

- SimpleAciUiLogServer
- SimpleAciUiLogServer.py
- acilogserv

The standalone script also allows you to set several options:

- -a or –apicip: The IP address of an APIC or an IP address on the same subnet as the APIC. This allows the standalone server to be able to print the correct IP address when it announces what IP address, port and location it is listening on if the server is multi-homed.
- -p or –port: The http port the server should listen on.
- -s or –sslport: The https port the server should listen on.
- -c or –cert: The server certificate for HTTPS connections.
- -l or –location: The local path that the server should look for, anything sent to the server outside of this location will result in the server returning a 404. The default is /apiinspector
- -r or –logrequests: This will cause the server to log a message about the POST request to sys.stderr, the default is False, possible values are True and False.
- -d or –delete-imdata: Strip out the im_data and other info at the im_data level from the payloads and responses.
- -n or –nice-output: Pretty print the payloads and responses.

- -i or –indent: Number of spaces to indent when pretty printing.

When the module is run as a standalone script it simply logs the messages to sys.stdout using the standard logging module.

CHAPTER 2

APIC Configuration

Once the server is running, you can start remote logging from the APIC UI by selecting “Start Remote Logging” from the ‘welcome, username’ menu in the top right corner of the APIC UI.

Then enter the URL the server is listening on:

If you need to disable the remote logging from the APIC, you can do so from the same menu and selecting ‘Stop Remote Logging.’

Note: If https is used to connect to the APIC, the server that is instantiated will also need to be able to accept https connections.

CHAPTER 3

Server Test

By convention the APIC does not use the GET method when communicating with the logging server. The APIC only uses POST to POST the log messages to the server. However, the servers provided by this module do offer a GET method to provide a means of testing them. For example it is possible to open a web browser and browse to the server that has been started. If the server is working a small message is provided about pointing the APIC to that server.

CHAPTER 4

Debugging

If things do not seem to be working, the first step should be to open the developer tools/javascript console for the browser and see if there are any errors being printed as you click on various items in the APIC GUI.

CHAPTER 5

Importing as a module

You can also import the module and use it as a server as part of another application. This provides you with flexibility as it allows the registration of callback functions for each HTTP method (GET, POST, DELETE, etc) found in the log message. From this, it is possible to do things like use the data from the log message for other purposes or filter out specific logs messages based on the HTTP method. The methods that the APIC uses are:

- GET
- POST
- EventChannelMessage
- undefined - NOTE: This is an APIC bug that is fixed in versions after 1.0(2*)

Example:

```
>>> from SimpleAciUiLogServer.SimpleAciUiLogServer import \
... SimpleAciUiLogServer
>>> import logging
>>>
>>> logging.basicConfig(level=logging.DEBUG)
>>> def GET(**kwargs):
...     logging.debug("Got a GET")
...
>>> def POST(**kwargs):
...     logging.debug("Kwargs/params: {0}".format(kwargs))
...
>>> server = SimpleAciUiLogServer(("", 8987), location='/apiinspector')
>>> server.register_function(GET)
>>> server.register_function(POST)
>>> server.serve_forever()
DEBUG:root:Got a GET
DEBUG:root:
    method: Event Channel Message
       url: N/A
   payload: N/A
    # objs: 0
  response: {"subscriptionId":["72057611234639895","72057611234640073"],
  "imdata":[{"fvTenant":{"attributes":{"childAction":"","dn":
  "uni/tn-mtimm-simple","modTs":"2015-01-23T23:04:28.838+00:00","rn":"",
  "status":"deleted"}}}]}

DEBUG:root:Kwargs/params: {'data': {'url':
'http://172.1.1.5/api/node/mo/uni.json', 'response': '{"imdata":[]}',
'preamble': '18:00:12 DEBUG - ', 'method': 'POST', 'payload': '{"polUni":{
```

```
"attributes":{"dn":"uni","status":"modified"},"children":[{"fvTenant":{
"attributes":{"dn":"uni/tn-mtimm-simple","status":"deleted"},"children":[]}
}]}}'}, 'layout': 'PatternLayout'}
```

Note: since there were no functions registered for the EventChannelMessage method, SimpleAciUiLogServer sent that data to the default dispatch method which logs a formatted message. However, both GET and POST have registered functions and they do different things than the default dispatch method.

It is also possible to override the _dispatch method to create your own dispatch logic, for example rather than dispatch based on method maybe you would like to dispatch based on subscription id.

5.1 Multi-threaded Servers

The SimpleAciUiLogServer class is single threaded. If many APIC's are going to be reporting into the same server, one transaction may block another until the first is complete. This scenario can be avoided using the ThreadingSimpleAciUiLogServer class. The ThreadingSimpleAciUiLogServer class provides a threaded server that can accept multiple connections at the same time. When using the ThreadingSimpleAciUiLogServer it is best to use the logging functionality from the Python standard library rather than print statements because the logging module is thread safe.

If you need to listen on multiple ports you will need to instantiate multiple, SimpleAciUiLogServer or ThreadingSimpleAciUiLogServers. This might be done to start up both a http and https server. The module provides its own serve_forever() method that dispatches to multiple server instances. Otherwise the servers own serve_forever() method is appropriate. The standalone script offers an example of doing this.

5.2 HTTPS TLS/SSL Support

To accept HTTPS connections the SimpleAciUiLogServer or the ThreadingSimpleAciUiLogServer classes can be instantiated with the cert parameter pointed at a file that contains the servers certificate. The module comes with an embedded self-signed certificate but use of this should be avoided in long-term production scenarios. A self-signed certificate can be created using openssl:

```
openssl req -new -x509 -keyout server.pem -out server.pem -days 36500 -nodes
```

When the cert parameter is passed to the class initializer and is not None, the socket is wrapped in ssl allowing the APIC to send https POST's to the server.

If you are using self-signed certificates, you will most likely need to accept the certificate as a security exception in your browser before the APIC can send data to it. This is usually a one-time configuration step and can most easily be accomplished by using your browser to browse to the server.

5.3 Available Class Variables

The servers provided inherit from a log dispatch class that offers some class variables to control how the server formats the log messages. Those variables are:

- prettyprint - Format the payload and responses so they are easier to read. The default is False.
- indent - When using prettyprint, how much indent should be used. The default is 4.
- strip_imdata - When printing responses, do not print the whole response, only print the contents of the im_data field. The default is False.

All three of these variables are booleans and should be set to True or False.

CHAPTER 6

Author and Acknowledgements

Written by Mike Timm (mtimm@cisco.com) Based on code written by Fredrik Lundh & Brian Quinlan.