Lab – Create a Host Inventory in Python

Objectives

Part 1: Use Postman to get a Network Hosts Inventory

Part 2: Use Python to get a Network Hosts Inventory

1. Background / Scenario

In a production network, the hosts on the network are constantly changing. From a security point of view it is very useful to know about these hosts and where they connect. The APIC-EM keeps track of the hosts that are connected on the network, and this information can be periodically collected and processed by Python. In this lab we will create a function that will display an inventory of hosts that can be processed by other Python programs.

First, we will learn about the requirements for making the host inventory request by using Postman. We will review the structure of the JSON that comprises the inventory. Next we will replicate the process in a Python program.

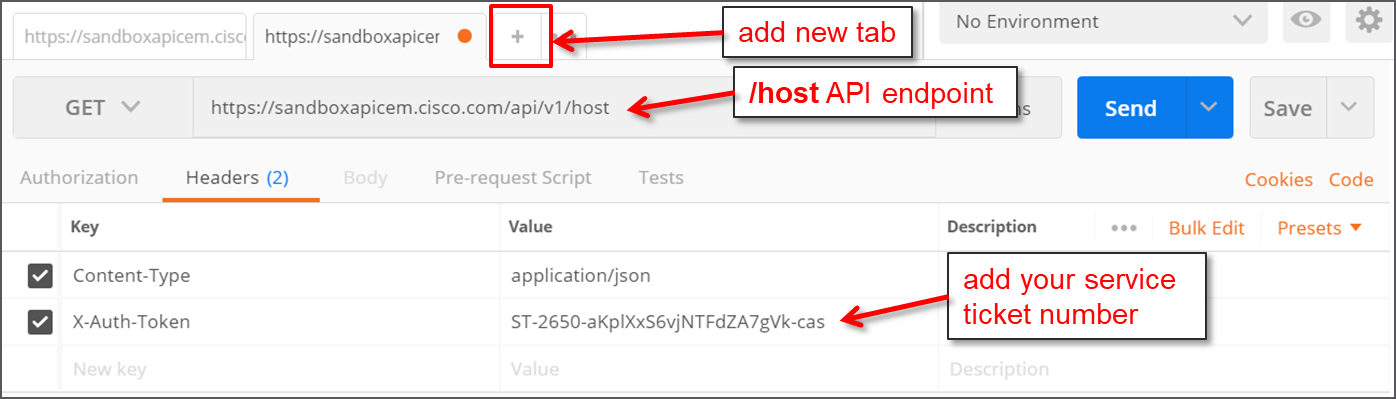
1. Required Resources

* Postman
* Python 3 with IDLE
* Python **requests** module
* Python **tabulate** module
* The functions file that you have created or the **apic\_em\_functions\_sol.py** file
* Access to the Internet

1. Use Postman to get a Network Hosts Inventory

Postman is an excellent tool for learning about an API before writing code for it. It is a good practice to visit the Swagger page for an endpoint to learn about the request requirements, and then try your request out in Postman. After verifying the requirements for accessing the API in Postman, and reviewing the JSON data that is returned, you can move on to coding the request in Python.

* + 1. Setup the Postman Request
       1. Create a new tab by clicking the + symbol that is next to the tab for the ticket request that you created in the previous lab. This part of the lab requires that the use of the ticket request tab.



* + - 1. In the new tab, enter the following information as shown in graphic:
         1. Request method: **GET**
         2. Endpoint URI: [**https://sandboxapicem.cisco.com/api/v1/host**](https://sandboxapicem.cisco.com/api/v1/host)
         3. Headers:

**Content-Type: application/json**

**X-Auth-Token:** <*leave this blank for now*>

* + - 1. Click the tab for the service ticket request. Run the request.
      2. Select the value of the service ticket, without quotes, from the response JSON.
      3. Return to the host inventory request tab.
      4. Paste the value of the service ticket into the Value field for the **X-Auth-Token** key.
      5. Send the request.

If the request is successful, you will see the body section of response populate with JSON data that represents the host inventory. If the response fails, look at the status value and try to determine where the error may be.

* + 1. Evaluate the Response
       1. The JSON that is returned for the host inventory is more extensive than what we have seen so far. Copy the JSON from Postman to JSON Viewer where the different levels can be collapsed and expanded by creating a Tree View. Click the **Tree Viewer** button to render the tree. The JSON Viewer URL is <https://codebeautify.org/jsonviewer>.
       2. Collapse all levels by clicking the second icon in the right hand corner of the Result window. Expand **object**, **array**, and **response**. The data from the API is contained in response. The number next to the response key indicate how many entries there are. Expand the **response** level and also the level marked **0**. Look at the keys that are assigned to level 0.

What does level 0 represent? How about levels 1 and 2?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Next to level 0, the number 14 indicates that there are 14 keys associated with this host. However, there are different numbers for the other hosts. Open each device and compare entries. Why is there a difference?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Use Python to get a Network Host Inventory
   * 1. Setup the code environment

As in the previous lab, we need to import modules.

* + - 1. Create a documentation header for you file with comments.
      2. Import the **requests**, **json**, and **sys** modules using import statements.
      3. Import tabulate and **my\_apic\_em\_functions** using **from… import \*** statements. We will use your **get\_ticket()** function in this lab. If necessary, import the **apic\_em\_functions\_sol** module if a working version of **my\_apic\_em\_functions** is unavailable.

import requests

import json

import sys

from tabulate import \*

from my\_apic\_em\_functions import \*

* + 1. Build request components
       1. Create the variable **post\_url** and assign a string containing the URI of the APIC-EM **/host** endpoint. The URI is [**http://sandoxapicem.cisco.com/api/v1/host**](http://sandoxapicem.cisco.com/api/v1/host). Look at the work you did for the last lab if you need help.
       2. Create the variable **ticket** and assign to it the value returned by the **get\_ticket()** function.

ticket = get\_ticket()

* + - 1. Create the **headers** dictionary and assign it to the **headers** variable. The header needs to include the content type and authentication token as it did in Postman. We will use the ticket variable that was created in the last step as the service ticket value. This is created as a two key dictionary. Be careful with the punctuation.

headers = {'content-type':'application-json','X-Auth-Token':ticket}

* + 1. Make the Request and Handle Errors

This section of the code uses a **try: except:** structure. This is a very handy structure in Python that captures errors and allows them to be handled. The code in the **try:** section runs, and if an error is encountered from it, the code in the **except**: section will run. This can help avoid program crashes. We use it here in case the request fails.

* + - 1. Type **try:** and press enter. The next line should automatically indent. If it does not, insert four spaces before the next line.
      2. Create the variable resp, and assign to it the results of request. We will use the **requests.get()** method to make the request. We supply to it the URI and headers variables that we created above as arguments as well as a couple of necessary settings.

resp = requests.get(post\_url,headers=headers,params='',verify=False)

* + - 1. Next we print the status of the request. This will let us know what the status of our request is. Since **resp** is a **requests** object, it has a property called **status\_code**, which is an integer. We want to print a string with some explanatory text followed by the status code. In order to do that we need to convert the status code integer to a string by using the Python **str()** function.

print('Status of /host request: ' = str(resp.status\_code))

* + - 1. Now we need to complete the **except:** part of the code. Type **except:**. It should not be indented. In other words, it level of ident should match that of the **try:** statement.
      2. Now enter the code that should execute if an error is encountered. It must be indented to the same level as the code in the **except:** portion of the structure. In this case, we only print a message and exit the program. Much more could be done to diagnose what errors occurred and to handle them. Enter the following statements:

print ('Something is wrong with GET /host request!')

sys.exit()

The entire block of code should look like this:

try:

resp = requests.get(post\_url,headers=headers,params='',verify = False)

response\_json = resp.json() # Get the json-encoded content from response

print ('Status of /host request: ',str(resp.status\_code))

except:

print ('Something is wrong with GET /host request!')

sys.exit()

* + 1. Evaluate the Response

We now need to extract the information that we want from the JSON for the hosts. We would like to display the type and IP address for each host. We would also like to display an ordinal number next to each host. Finally, we would like to display the host information is a nicely formatted table.

To do this, we create a **for:** loop. The **for:** loop will iterate through the list of hosts and extract the value for the two dictionary keys that we want. They are the **hostType** and **hostIP** keys of the **response** dictionary. We create a list variable called **host\_list** that will contain the ordinal number, host type, and host IP address. Each host will be appended to the list as a separate line as the loop executes. Execution of loop stops when there are no more items under the **response** dictionary key.

The ordinal number of the host is not present in the JSON data that we are working with. We will create a separate variable, assign a value of 0 and then increment it as the loop repeats.

* + - 1. First initialize the two variables that we require. The first is a list variable called host\_list. The second is an integer variable that will hold the ordinal.

host\_list = []

i = 0

* + - 1. Now create the **for:** loop. The loop will iterate over every item in the response key of the **response\_json** variable. Each time that loop iterates, the **item** variable takes on the value of the JSON for that item in response, which corresponds to each host. The statement that does this is:

for item in response\_json['response']:

* + - 1. The next lines of code will execute as the **for:** loop runs. They should be indented. First, we increment the ordinal number variable **i**. Then, we append the values from the JSON to the host\_list variable.

i += 1

The key to understanding the next line of code is that the variable **item** takes on the value of the keys under the response key with every iteration. In essence, **item** holds all the information in the JSON for device 0 in the first iteration, device 1 in the next, etc. This means that **item** has the **hostType** and **hostIP** keys within it. The following line of code appends the ordinal variable, **hostType**, and **hostIP** values to the **host\_list** variable. As the loop iterates it does this for each host.

host\_list.append([[i,item['hostType'],item['hostIp']])

The complete **for:** loop should look like this:

for item in response\_json['response']:

i+=1

host\_list.append([i,item['hostType'],item['hostIp']])

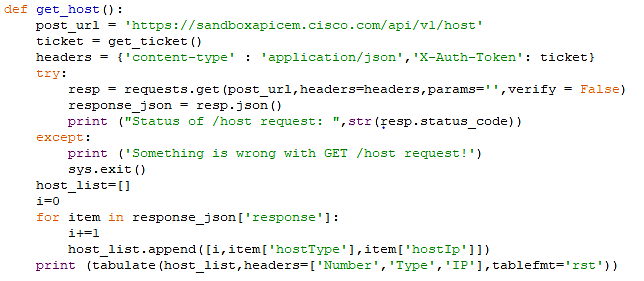
* + - 1. Finally, we print the table of hosts. We use the tabulate module to do this. The tabulate module will take the **host\_list** variable, a list of headers for the columns that will be printed, and setting for the table format.

print(tabulate(host\_list,headers=['Number','Type','IP'],tablefmt='rst'))

* + - 1. Save your code file as **get\_host.py**. Run the code. Investigate any errors that may occur. Look for mismatched paired symbols like (), [], and ' '.
    1. Create the function

You will now add this code to the functions file that contains your **get\_ticket()** function.

* + - 1. Select all the code and copy.
      2. Return to your functions file and paste the code in just below the import statements in the function file.
      3. Delete the statement that imports functions from this file. You can't import a file into itself.
      4. Define the function with the **def get\_host()** statement.
      5. Select all of the lines below the definition statement and select Indent Region from the IDLE format menu. Everything should be imported an additional four spaces or so. This function does not require a **return** statement.
      6. Save the file. Your function should look like this:



* + - 1. To test your function, run the functions file. In the shell that opens, type **get\_host()** at the prompt. The function should run as the program did.

You have now coded a request to the host inventory endpoint and reused your **get\_ticket** code as a function. In the next lab, you will modify the **get\_host** code to display a table of network devices.