ISE DNAC Network Devices Consistency Check

Consistency Checking Procedure

ISE

1. Read the username, password and IP address of ISE instance from ‘input.txt’.
2. Encode the username and password for the request to authorize access to the ISE instance.

﻿Encoding can be done with following steps:

encodedvalue=un+":"+pw

b64Val = base64.b64encode(encodedvalue.encode('UTF-8')).decode('utf-8')

Get the Policy matrix from ISE

STEP 1:

Call the API <https://ip/ers/config/egressmatrixcell?simple=yes&page=1> to get ISE EgressMatrixCell ID’s.

Code to call the API is as follows:

﻿ids=[]

repos=[]

try: r=requests.get('https://'+ip+'/ers/config/egressmatrixcell?simple=yes&page=1',headers={"Authorization": "Basic %s" % b64Val,"Content-Type": "application/json","Accept": "application/json"}, verify=False)

#checking for correct username and password

if r.status\_code!=200:

print("Please check the data you entered again")

return("error",r.status\_code)

json\_data = json.loads(r.text)

repos.append(json\_data)

#checking for next page

try:

while json\_data["SearchResult"]["nextPage"]['rel'] == 'next':

url=json\_data["SearchResult"]["nextPage"]['href']

r=requests.get(url,headers={"Authorization": "Basic %s" % b64Val,"Content-Type": "application/json","Accept": "application/json"}, verify=False)

json\_data = json.loads(r.text)

try:

x=json\_data["SearchResult"]["nextPage"]

except:

print("")

repos.append(json\_data)

except:

print("")

for i in repos:

x=i['SearchResult']['resources']

for element in range(len(x)):

ids.append(x[element]['id'])

#Error handling

r.raise\_for\_status()

except requests.exceptions.Timeout as errt:

print ("Timeout Error:",errt)

sys.exit(1)

except requests.exceptions.ConnectionError as errc:

print ("Error Connecting:",errc)

sys.exit(1)

except requests.exceptions.HTTPError as errh:

print ("Http Error:",errh)

sys.exit(1)

except requests.exceptions.RequestException as err:

print ("Oops: Something Else",err)

sys.exit(1)

Response from API call is as follows:

{

"SearchResult" : {

"total" : 27,

"resources" : [ {

"id" : "62012531-1614-11e9-b240-024244f34a4a",

"name" : "57B171-ciscogithub",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/62012531-1614-11e9-b240-024244f34a4a",

"type" : "application/xml"

}

}, {

"id" : "5cc2f411-1617-11e9-b240-024244f34a4a",

"name" : "6XMFLK-ciscogithub",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/5cc2f411-1617-11e9-b240-024244f34a4a",

"type" : "application/xml"

}

}, {

"id" : "92c1a900-8c01-11e6-996c-525400b48521",

"name" : "ANY-ANY",

"description" : "Default egress rule",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/92c1a900-8c01-11e6-996c-525400b48521",

"type" : "application/xml"

}

}, {

"id" : "15f542b1-1615-11e9-b240-024244f34a4a",

"name" : "D8I7FN-ciscogithub",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/15f542b1-1615-11e9-b240-024244f34a4a",

"type" : "application/xml"

}

}, {

"id" : "25e7f181-1616-11e9-b240-024244f34a4a",

"name" : "EDYWRD-ciscogithub",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/25e7f181-1616-11e9-b240-024244f34a4a",

"type" : "application/xml"

}

}, {

"id" : "2424d9a0-15a6-11e9-b240-024244f34a4a",

"name" : "HNC84D-ciscogithub",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/2424d9a0-15a6-11e9-b240-024244f34a4a",

"type" : "application/xml"

}

}, {

"id" : "af001141-3ff7-11e9-b240-024244f34a4a",

"name" : "IMZQV8-ciscogithub",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/af001141-3ff7-11e9-b240-024244f34a4a",

"type" : "application/xml"

}

}, {

"id" : "ce599480-3ff7-11e9-b240-024244f34a4a",

"name" : "IMZQV8-ciscogithub",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/ce599480-3ff7-11e9-b240-024244f34a4a",

"type" : "application/xml"

}

}, {

"id" : "c42baf20-3ff7-11e9-b240-024244f34a4a",

"name" : "IMZQV8-ciscogithub",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/c42baf20-3ff7-11e9-b240-024244f34a4a",

"type" : "application/xml"

}

}, {

"id" : "b9cb6f21-3ff7-11e9-b240-024244f34a4a",

"name" : "IMZQV8-ciscogithub",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/b9cb6f21-3ff7-11e9-b240-024244f34a4a",

"type" : "application/xml"

}

}, {

"id" : "a0d25e50-15a1-11e9-b240-024244f34a4a",

"name" : "L0A5E7-ciscogithub",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/a0d25e50-15a1-11e9-b240-024244f34a4a",

"type" : "application/xml"

}

}, {

"id" : "162d90c0-1615-11e9-b240-024244f34a4a",

"name" : "ciscogithub-D8I7FN",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/162d90c0-1615-11e9-b240-024244f34a4a",

"type" : "application/xml"

}

}, {

"id" : "d9fe8e11-38e7-11e9-b240-024244f34a4a",

"name" : "ciscogithub-nvm\_sgt",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/d9fe8e11-38e7-11e9-b240-024244f34a4a",

"type" : "application/xml"

}

}, {

"id" : "2618ec91-1616-11e9-b240-024244f34a4a",

"name" : "ciscogithub-EDYWRD",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/2618ec91-1616-11e9-b240-024244f34a4a",

"type" : "application/xml"

}

}, {

"id" : "624d4960-1614-11e9-b240-024244f34a4a",

"name" : "ciscogithub-57B171",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/624d4960-1614-11e9-b240-024244f34a4a",

"type" : "application/xml"

}

}, {

"id" : "a12a6961-15a1-11e9-b240-024244f34a4a",

"name" : "ciscogithub-L0A5E7",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/a12a6961-15a1-11e9-b240-024244f34a4a",

"type" : "application/xml"

}

}, {

"id" : "2473e401-15a6-11e9-b240-024244f34a4a",

"name" : "ciscogithub-HNC84D",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/2473e401-15a6-11e9-b240-024244f34a4a",

"type" : "application/xml"

}

}, {

"id" : "5cffaef1-1617-11e9-b240-024244f34a4a",

"name" : "ciscogithub-6XMFLK",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/5cffaef1-1617-11e9-b240-024244f34a4a",

"type" : "application/xml"

}

}, {

"id" : "1a162e20-157c-11e9-b240-024244f34a4a",

"name" : "ciscogithub-test\_sgt",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/1a162e20-157c-11e9-b240-024244f34a4a",

"type" : "application/xml"

}

}, {

"id" : "498d1ec0-36b0-11e9-b240-024244f34a4a",

"name" : "ciscogithub-teacher",

"description" : "APIC-EM pushed ",

"link" : {

"rel" : "self",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell/498d1ec0-36b0-11e9-b240-024244f34a4a",

"type" : "application/xml"

}

} ],

"nextPage" : {

"rel" : "next",

"href" : "https://10.104.55.38:9060/ers/config/egressmatrixcell?simple=yes&page=2",

"type" : "application/xml"

}

}

}

If there is a next page as highlighted in the response copy the “href” into url and call the API again. Append json format of all responses into a list names repos. Extract the “id” from each response and append into a list named ids.

STEP 2:

For each EgressMatrixCell ID call the API <https://ip/ers/config/egressmatrixcell/id> to get contents of the cell.

Code to call the API is as follows:

responses=[]

for element in ids:

try:

r=requests.get('https://'+ip+'/ers/config/egressmatrixcell/'+element,headers={"Authorization": "Basic %s" % b64Val,"Content-Type": "application/json","Accept": "application/json"}, verify=False)

json\_data = json.loads(r.text)

responses.append(json\_data)

#error handling

Append the responses corresponding to each Cell ID into a list names responses.

The contents of one EgressMatrixCell is as follows:

{'EgressMatrixCell': {'id': '62012531-1614-11e9-b240-024244f34a4a', 'name': '57B171-ciscogithub', 'sourceSgtId': '61be76e0-1614-11e9-b240-024244f34a4a', 'destinationSgtId': 'd073e690-1495-11e9-b240-024244f34a4a', 'matrixCellStatus': 'DISABLED', 'defaultRule': 'DENY\_IP', 'sgacls': ['92919850-8c01-11e6-996c-525400b48521'], 'link': {'rel': 'self', 'href': 'https://10.104.55.38:9060/ers/config/egressmatrixcell/62012531-1614-11e9-b240-024244f34a4a', 'type': 'application/xml'}}}

STEP 3:

From the responses extract ‘sourceSgtId’,’destinationSgtId’,’sgacls’ and append into sgtid, dgtid and sgaclid lists.

The code is as follows:

sgtid=[]

dgtid=[]

sgaclid=[]

for i in range(len(responses)):

sgtid.append(responses[i]['EgressMatrixCell']['sourceSgtId'])

dgtid.append(responses[i]['EgressMatrixCell']['destinationSgtId'])

sgaclid.append(responses[i]['EgressMatrixCell']['sgacls'])

STEP 4:

For each sgtid get the sgtresponse by calling the API <https://ip/ers/config/sgt/sgtid> and append the responses into a list sgtresp.

Similarly for each dgtid get the dgtresponse by calling the API <https://ip/ers/config/sgt/dgtid> and append the responses into a list dgtresp.

Code is as follows:

for i in range(len(gtid)):

try:

r=requests.get('https://'+ip+'/ers/config/sgt/'+gtid[i],headers={"Authorization": "Basic %s" % b64Val,"Content-Type": "application/json","Accept": "application/json"}, verify=False)

#print(r.text)

json\_data = json.loads(r.text)

gtresp.append(json\_data)

#error handling

Extract the sgt and dgt from sgtresponse and dgtresponse respectively.

dgtval=[]

for i in range(len(dgtresp)):

dgtval.append(dgtresp[i]['Sgt']['value'])

sgtval=[]

for i in range(len(sgtresp)):

sgtval.append(sgtresp[i]['Sgt']['value'])

Sgtresponse for one sgt-id is as follows:

{'Sgt': {'id': '92bb1950-8c01-11e6-996c-525400b48521', 'name': 'ANY', 'description': 'Any Security Group', 'value': 65535, 'generationId': '5', 'propogateToApic': False, 'link': {'rel': 'self', 'href': 'https://10.197.241.91:9060/ers/config/sgt/92bb1950-8c01-11e6-996c-525400b48521', 'type': 'application/xml'}}}

Dgtresponse for one dgt-id is as follows:

{'Sgt': {'id': 'd073e690-1495-11e9-b240-024244f34a4a', 'name': 'ciscogithub', 'description': 'Ciscogithub server', 'value': 20, 'generationId': '25', 'propogateToApic': True, 'link': {'rel': 'self', 'href': 'https://10.104.55.38:9060/ers/config/sgt/d073e690-1495-11e9-b240-024244f34a4a', 'type': 'application/xml'}}}

STEP 5:

For each sgaclid get the sgaclresponse by calling the API <https://ip/ers/config/sgacl/sgaclid>

Code is as follows:

for i in sgaclid:

responses=[]

for l in i:

try:

r= requests.get('https://'+ip+'/ers/config/sgacl/'+l,headers={"Authorization": "Basic %s" % b64Val,"Content-Type": "application/json","Accept": "application/json"}, verify=False)

json\_data = json.loads(r.text)

responses.append(json\_data)

#error handling

sgaclresp.append(responses)

Append all the responses into a list named sgaclresp. From the sgaclresp extract sgaclval and aclcontent.

for i in sgaclresp:

sgaclval=[]

aclcont=[]

for j in i:

sgaclval.append(j['Sgacl']['name'])

x=j['Sgacl']['aclcontent']

k=x.split("\n")

if len(k)==1:

for z in k:

aclcont.append(z)

else:

aclcont.append(k)

sgaclvallist.append(sgaclval)

aclcontlist.append(aclcont)

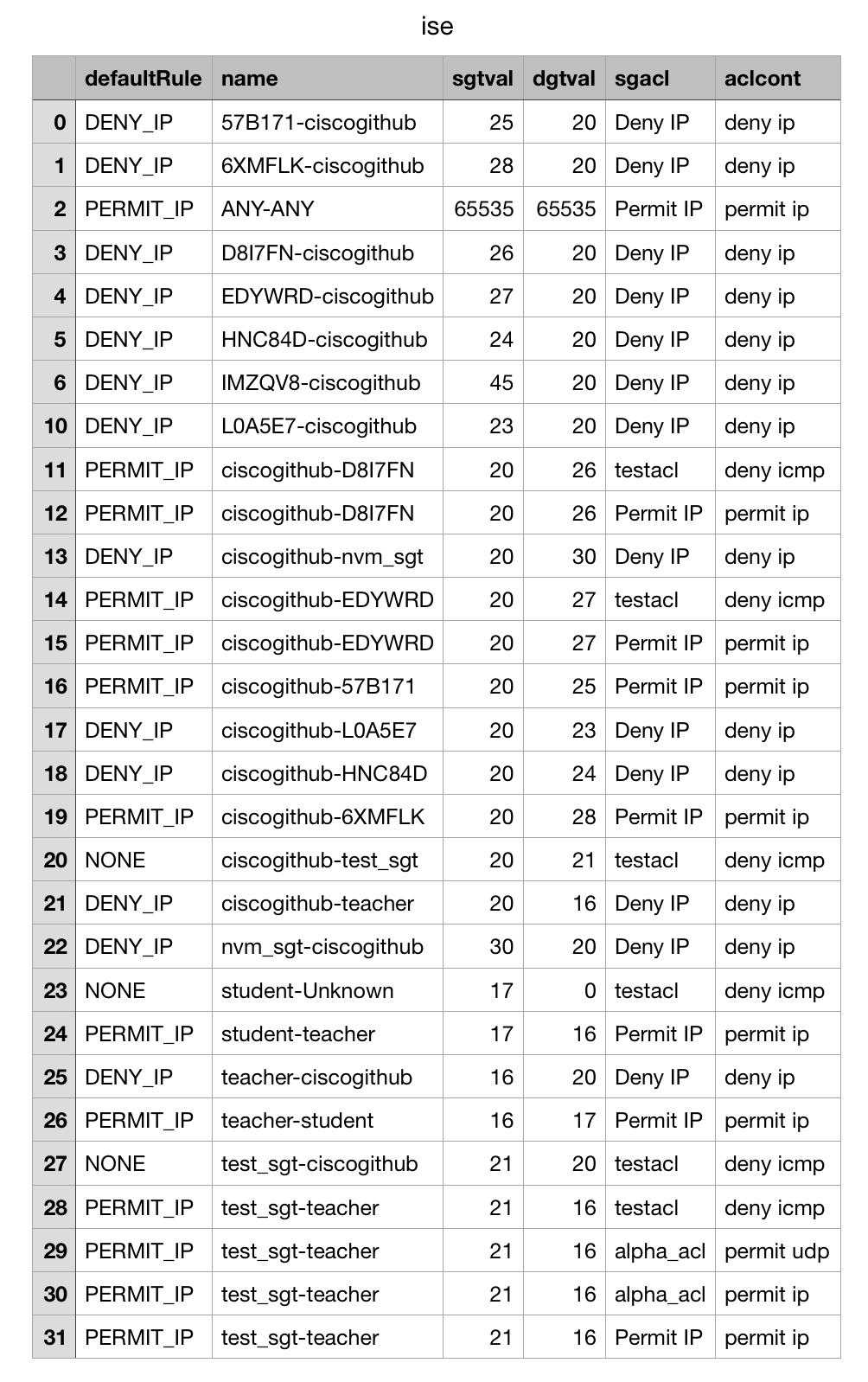
The sgaclresp for one sgaclid is as follows:

{'Sgacl': {'id': '92919850-8c01-11e6-996c-525400b48521', 'name': 'Deny IP', 'description': 'Deny IP SGACL', 'generationId': '0', 'aclcontent': 'deny ip', 'link': {'rel': 'self', 'href': 'https://10.104.55.38:9060/ers/config/sgacl/92919850-8c01-11e6-996c-525400b48521', 'type': 'application/xml'}}}

Extract ‘name’ and ‘aclcontent’ and append it to list of sgacl’s and to list of aclcont’s respectively.

['Deny IP'] ['deny ip']

Create a Data frame for ISE data as shown below



1. Delete duplicate rows in data frame if any.
2. Check for “ANY-ANY” name. If present sgacl corresponding to “ANY-ANY” is the default permission.

7. Check if 0 (unknown) value is present in dgt column and assign unknown value to 1 if present.

DNAC-Network Devices

1. Read username, password and IP Address of DNAC instance.
2. Encode the username and password for the request to authorize access to the DNAC instance.

﻿Encoding can be done with following steps:

encodedvalue=un+":"+pw

b64Val = base64.b64encode(encodedvalue.encode('UTF-8')).decode('utf-8')

1. Call the API <https://ip/api/system/v1/auth/login> to obtain an access token. The token obtained using this API is required to be set as value to the X-Auth-Token HTTP Header for all API calls to DNAC.

Extract the cookie from API response using following code:

a=r.headers['Set-Cookie'].split(";")

b=a[0].split("=")

c=b[1]

cookie = {'X-JWT-ACCESS-TOKEN':c}

1. Call the API ﻿<https://ip/api/v1/network-device/count> to get the count of devices connected to DNAC.
2. Call the API ﻿[https://ip/api/v1/network-device](https://ip/api/v1/network-devicet) to get the details of network devices connected to DNAC.
3. If the devices are Wireless Controllers or Access Points skip them.
4. If the “errorCode” of the device is ‘DEV-UNREACHED’ skip it.
5. Extract the instanceUuid, type and managementIpAddress of the devices.

The response from API [https://ip/api/v1/network-device](https://ip/api/v1/network-devicet) is as below:

{'type': 'Cisco Catalyst 9300 Switch', 'family': 'Switches and Hubs', 'errorCode': None, 'location': None, 'role': 'ACCESS', 'lastUpdateTime': 1553591737180, 'hostname': '9300BorderCP.vp.com', 'macAddress': '74:86:0b:c5:e0:00', 'softwareType': 'IOS-XE', 'softwareVersion': '16.9.2', 'serialNumber': 'FCW2133L0QE', 'collectionInterval': 'Global Default', 'inventoryStatusDetail': '<status><general code="SUCCESS"/></status>', 'upTime': '77 days, 2:34:44.49', 'roleSource': 'AUTO', 'apManagerInterfaceIp': '', 'associatedWlcIp': '', 'bootDateTime': '2018-08-20 05:19:27', 'collectionStatus': 'Managed', 'errorDescription': None, 'interfaceCount': '70', 'lastUpdated': '2019-03-26 09:15:37', 'lineCardCount': '2', 'lineCardId': 'a5833329-fbcf-48cc-a624-8f8561367d94, b1b4561f-b5ae-47d3-836c-2990ab75a3b5', 'locationName': None, 'managementIpAddress': '30.1.0.91', 'memorySize': '1451820200', 'platformId': 'C9300-48U', 'reachabilityFailureReason': '', 'reachabilityStatus': 'Reachable', 'series': 'Cisco Catalyst 9300 Series Switches', 'snmpContact': '', 'snmpLocation': '', 'tagCount': '0', 'tunnelUdpPort': None, 'waasDeviceMode': None, 'instanceTenantId': 'SYS0', 'instanceUuid': 'fb54251b-751b-431a-9623-c16e667cc18b', 'id': 'fb54251b-751b-431a-9623-c16e667cc18b'}

This device is skipped because it is access point.

{'type': 'Cisco 1810W Unified Access Point', 'family': 'Unified AP', 'errorCode': 'null', 'location': None, 'role': 'ACCESS', 'lastUpdateTime': 1553592280073, 'hostname': 'AP00A2.890E.5420', 'macAddress': '00:a2:89:0f:b8:20'}

This device is skipped because it is wireless controllers.

{'type': 'Cisco 5520 Series Wireless Controllers', 'family': 'Wireless Controller', 'errorCode': None, 'location': None, 'role': 'ACCESS', 'lastUpdateTime': 1553592280073, 'hostname': 'WLCVP', 'macAddress': '00:d7:8f:7e:72:db'}

This device is skipped because its error code is DEV-UNREACHED.

{'type': 'Cisco Catalyst 9300 Switch', 'family': 'Switches and Hubs', 'errorCode': 'DEV-UNREACHED', 'location': None, 'role': 'ACCESS', 'lastUpdateTime': 1553591737180, 'hostname': '9300BorderCP.vp.com', 'macAddress': '74:86:0b:c5:e0:00'}

From each device we get the PAC and SXP information

Using the body = {"commands": ["show cts sxp connections","show cts pac"],

"description": "string",

"deviceUuids": [devid],

"name": "string",

"timeout": 0 },

Call the API <https://ip/api/v1/network-device-poller/cli/read-request> to get task-id.

To extract the task-id:

pattern = re.compile(r'taskId":"([a-zA-z0-9-]+)"')

match=pattern.findall(r.text)

task\_id = match[0]

Response:

POLL: {"response":{"taskId":"b19b9fd2-06a2-44af-85f8-0b511990e106","url":"/api/v1/task/b19b9fd2-06a2-44af-85f8-0b511990e106"},"version":"1.0"}

Using the task-id call the API <https://ip/api/v1/task/task_id> to get file-id.

To extract the file-id:

pattern=re.compile(r'fileId\\":\\"([a-zA-Z0-9-]+)\\')

match=pattern.findall(r.text)

file\_id=match[0]

Response:

TASK: {"response":{"version":1553593660017,"startTime":1553593658858,"endTime":1553593660017,"serviceType":"Command Runner Service","username":"admin","isError":false,"lastUpdate":1553593660017,"progress":"{\"fileId\":\"741e0b15-66c8-4ae8-b56b-b47302fdee7d\"}","rootId":"b19b9fd2-06a2-44af-85f8-0b511990e106","instanceTenantId":"5b3e23262fce6e008b1cce4a","id":"b19b9fd2-06a2-44af-85f8-0b511990e106"},"version":"1.0"}

Give 2 seconds of delay so that DNAC returns proper file-id. Using the file-id call the API <https://ip/api/v1/file/file_id> to get pac and sxp information.

The response is as follows:

PAC AND SXP [{ "deviceUuid":"fb54251b-751b-431a-9623-c16e667cc18b" ,"commandResponses":{"SUCCESS":{"show cts pac":"show cts pac\nAID: F90022D65870E4005C5D7BF7392786AC\nPAC-Info:\n PAC-type \u003d Cisco Trustsec\n AID: F90022D65870E4005C5D7BF7392786AC\n I-ID: FCW2133L0QE\n A-ID-Info: Identity Services Engine\n Credential Lifetime: 06:41:20 UTC Mon Apr 08 2019\nPAC-Opaque: 000200B80003000100040010F90022D65870E4005C5D7BF7392786AC0006009C00030100BD2B2AF4A427C8F1F28B1D427B2CA478000000135C2C3B2800093A806A2F25A6FDCFAB135EFA7A170E8349091A58BE9E9E451BDE6B97C9F189539C1BE3FDED81D33B17FC64C1CDC9E249F932929D486AEF2846162F97013F8242D993F23511E7ED7464775DB073D7E69659B9DD5EEFFE70E48959DFFA02BEE81237BCB631798701FC91858119394381D4621FCA947659B4A706A3F5C7D317\nRefresh timer is set for 1w4d\n\n9300BorderCP#","show cts sxp connections":"show cts sxp connections\n SXP : Disabled\n Highest Version Supported: 4\n Default Password : Not Set\n Default Source IP: Not Set\nConnection retry open period: 120 secs\nReconcile period: 120 secs\nRetry open timer is not running\nPeer-Sequence traverse limit for export: Not Set\nPeer-Sequence traverse limit for import: Not Set\n\nThere are no SXP Connections.\n\n9300BorderCP#"},"FAILURE":{},"BLACKLISTED":{}}}]

Extract the PAC Credential Lifetime information from the response using the code below:

l=output.find('Credential Lifetime')

if l==-1:

pacinfo.update({"device%s"%count:"No Information found"}) #credential lifetime information is missing

else:

output=output[l:]

out1=output.split("\\n")

pacinfo.update({"device%s"%count:out1[0]})

Extract the SXP information from the response using the code below:

pattern = re.compile(r'\\n\s(SXP)\s+:\s(\w+)\\n')

match=pattern.findall(output)

for el in match:

sxp="{0} : {1}".format(el[0],el[1])

sxpinfo.update({"device%s"%count:sxp})

For each Device check the default permissions if “ANY-ANY” name was found in ISE data frame

Using the body = {"commands": ['sh cts role-based permission default'],

"description": "string",

"deviceUuids": [devid],

"name": "string",

"timeout": 0 },

Call the API <https://ip/api/v1/network-device-poller/cli/read-request> to get task-id.

Using the task-id call the API <https://ip/api/v1/task/task_id> to get file-id.

Give 2 seconds of delay so that DNAC returns proper file-id. Using the file-id call the API <https://ip/api/v1/file/file_id> to get default permissions information.

The response is as below:

1. [{ "deviceUuid":"fb54251b-751b-431a-9623-c16e667cc18b" ,"commandResponses":{"SUCCESS":{"sh cts role-based permission default":"sh cts role-based permission default\nRBACL Monitor All for Dynamic Policies : FALSE\nRBACL Monitor All for Configured Policies : FALSE\n\n9300BorderCP#"},"FAILURE":{},"BLACKLISTED":{}}}]
2. [{ "deviceUuid":"0c73c17a-1f25-46dd-b2ad-4cceb319896f" ,"commandResponses":{"SUCCESS":{"sh cts role-based permission default":"sh cts role-based permission default\nIPv4 Role-based permissions default:\n\tPermit IP-00\nRBACL Monitor All for Dynamic Policies : FALSE\nRBACL Monitor All for Configured Policies : FALSE\n\n9300MSEdge#"},"FAILURE":{},"BLACKLISTED":{}}}]

Extract the default permission from the response using code below:

res = re.findall(r'IPv4 Role-based permissions default:\\n\\t([a-zA-z \_]+)-..',r.text)

Compare the default permission extracted from the device with default permission from ISE.

if res==defperm:

defaultperm.update({"device%s"%count:str(res)+" matches"})

else:

defaultperm.update({"device%s"%count:str(res)+" does not match"})

### 

### From each device we get the corresponding sgt mapping

Using the body = {"commands": ["sh vrf"],

"description": "string",

"deviceUuids": [devid],

"name": "string",

"timeout": 0 },

Call the API <https://ip/api/v1/network-device-poller/cli/read-request> to get task-id.

Using the task-id call the API <https://ip/api/v1/task/task_id> to get file-id.

Give 2 seconds of delay so that DNAC returns proper file-id. Using the file-id call the API <https://ip/api/v1/file/file_id> to get vrfnames.

The response is as follows:

VRFRESPONSE: 'sh vrf\n Name Default RD Protocols Interfaces\n CriticalVandD 1:4102 ipv4 Lo2047\n LI0.4102\n Guest 1:4101 ipv4 Lo1026\n LI0.4101\n Mgmt-vrf <not set> ipv4,ipv6 Gi0/0\n Organization 1:4100 ipv4 Vl3005\n Lo1023\n LI0.4100\n School 1:4099 ipv4 Lo4099\n Vl3006\n Lo1025\n Lo1021\n Lo1027\n Tu0\n LI0.4099\nBorder2#'

Extracting vrf names from the response using the code below:

output = json\_data[0]['commandResponses']['SUCCESS']['sh vrf']

vrfnames = re.findall("\n\s\s([\w+-]+)\s+", output, re.M | re.I

vrfnames = vrfnames[1:]

For every vrfname:

Using the body = {"commands": ["sh cts role-based sgt-map vrf ‘vrfname’ all" ],

"description": "string",

"deviceUuids": [devid],

"name": "string",

"timeout": 0 },

Call the API <https://ip/api/v1/network-device-poller/cli/read-request> to get task-id.

Using the task-id call the API <https://ip/api/v1/task/task_id> to get file-id.

Give 2 seconds of delay so that DNAC returns proper file-id. Using the file-id call the API <https://ip/api/v1/file/file_id> to get sgt’s.

For global sgt response:

Using the body = {"commands": ["sh cts role-based sgt-map all"],

"description": "string",

"deviceUuids": [devid],

"name": "string",

"timeout": 0 },

Call the API <https://ip/api/v1/network-device-poller/cli/read-request> to get task-id.

Using the task-id call the API <https://ip/api/v1/task/task_id> to get file-id.

Give 2 seconds of delay so that DNAC returns proper file-id. Using the file-id call the API <https://ip/api/v1/file/file_id> to get sgts.

The responses are as follows:

SGTRESP: [{ "deviceUuid":"87e7a2ee-3773-4361-bf12-704c35c6fbeb" ,"commandResponses":{"SUCCESS":{"sh cts role-based sgt-map vrf CriticalVandD all":"sh cts role-based sgt-map vrf CriticalVandD all\n%IPv6 protocol is not enabled in VRF CriticalVandD\nMSMRplusBorder#"},"FAILURE":{},"BLACKLISTED":{}}}]

SGTRESP: [{ "deviceUuid":"87e7a2ee-3773-4361-bf12-704c35c6fbeb" ,"commandResponses":{"SUCCESS":{"sh cts role-based sgt-map vrf Guest all":"sh cts role-based sgt-map vrf Guest all\n%IPv6 protocol is not enabled in VRF Guest\nMSMRplusBorder#"},"FAILURE":{},"BLACKLISTED":{}}}]

SGTRESP: [{ "deviceUuid":"87e7a2ee-3773-4361-bf12-704c35c6fbeb" ,"commandResponses":{"SUCCESS":{"sh cts role-based sgt-map vrf Mgmt-vrf all":"sh cts role-based sgt-map vrf Mgmt-vrf all\nMSMRplusBorder#"},"FAILURE":{},"BLACKLISTED":{}}}]

SGTRESP: [{ "deviceUuid":"87e7a2ee-3773-4361-bf12-704c35c6fbeb" ,"commandResponses":{"SUCCESS":{"sh cts role-based sgt-map vrf Organization all":"sh cts role-based sgt-map vrf Organization all\n%IPv6 protocol is not enabled in VRF Organization\nMSMRplusBorder#"},"FAILURE":{},"BLACKLISTED":{}}}]

SGTRESP: [{ "deviceUuid":"87e7a2ee-3773-4361-bf12-704c35c6fbeb" ,"commandResponses":{"SUCCESS":{"sh cts role-based sgt-map vrf School all":"sh cts role-based sgt-map vrf School all\n%IPv6 protocol is not enabled in VRF School\nActive IPv4-SGT Bindings Information\n\nIP Address SGT Source\n\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\n31.31.31.4 17 SXP\n31.31.31.6 16 SXP\n64.103.217.48 20 SXP\n\nIP-SGT Active Bindings Summary\n\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\u003d\nTotal number of SXP bindings \u003d 3\nTotal number of active bindings \u003d 3\n\nMSMRplusBorder#"},"FAILURE":{},"BLACKLISTED":{}}}]

Extract the sgt’s and their corresponding IP addresses.

[('31.31.31.4', '17'), ('31.31.31.6', '16'), ('64.103.217.48', '20')]

SGTRESP: [{ "deviceUuid":"87e7a2ee-3773-4361-bf12-704c35c6fbeb" ,"commandResponses":{"SUCCESS":{"sh cts role-based sgt-map all":"sh cts role-based sgt-map all\nMSMRplusBorder#"},"FAILURE":{},"BLACKLISTED":{}}}]

Code to extract ip address and sgt’s from the response is as below:

for i in range(len(responses)):

if((responses[i].find('Active IPv4'))==-1): #if no sgt’s are found

sgtlist.append('')

sgtvrf.append('')

iplist.append('')

continue

sgt=[]

ips=[]

pattern=re.compile(r'\\n(\d\*\.\d\*\.\d\*\.\d\*/?\d\*?)\s+(\d+)')

match=pattern.findall(responses[i])

for i in match:

sgt.append(i[1])

ips.append(i[0])

sgtvrf.append(sgt)

sgt = list(dict.fromkeys(sgt)) #to remove duplicate sgts

sgtlist.append(sgt)

iplist.append(ips)

If ‘0’ dgt was found in ISE data frame add ‘0’ to sgtlist of the device.

if unknwnval == 1:

sgtlist.append(['0']) #Unknown sgt is 0

### For each of the SGT here get the configured policies

For each sgt in the sgt-list of the device:

Using the body = {"commands": ["sh cts role-based permission to sgt"],

"description": "string",

"deviceUuids": [devid],

"name": "string",

"timeout": 0 }

Call the API <https://ip/api/v1/network-device-poller/cli/read-request> to get task-id.

Using the task-id call the API <https://ip/api/v1/task/task_id> to get file-id.

Give 2 seconds of delay so that DNAC returns proper file-id. Using the file-id call the API <https://ip/api/v1/file/file_id> to get dgt’s, sgaclval and acl.

The response is as follows:

'[{ "deviceUuid":"87e7a2ee-3773-4361-bf12-704c35c6fbeb" ,"commandResponses":{"SUCCESS":{"sh cts role-based permission to 17":"sh cts role-based permission to 17\\nIPv4 Role-based permissions from group 16:teacher to group 17:student:\\n\\tPermit IP-00\\nRBACL Monitor All for Dynamic Policies : FALSE\\nRBACL Monitor All for Configured Policies : FALSE\\n\\nMSMRplusBorder#"},"FAILURE":{},"BLACKLISTED":{}}}]'

For sgt=17 dgt=16 sgaclval=Permit IP acl=Permit IP-00

Dgt, sgacl and acl can be extracted from the response using the code below:

res = re.findall(r'IPv4 Role-based permissions from group ([a-zA-Z\d]+)',r.text)

for n, i in enumerate(res):

if i == 'Unknown':

res[n] = '0'

if not res:

dgt.append('')

sgacl.append('')

acl.append('')

continue

dgt.append(res)

res2 = (r.text).split("IPv4")

res2.pop(0)

sgacl1=[]

acl1=[]

for i in res2:

res3 = re.findall(r'\(configured\):',i)#skip if its configured

if res3:

continue

res4 = re.findall(r'\\n\\t([\w\d -]+)',i)

res5 = re.findall(r'\\n\\t([\w ]+)',i)

sgacl1.append(res5)

acl1.append(res4)

sgacl.append(sgacl1)

acl.append(acl1)

Append the results into main dgt-list, sgacl-list and acl-list.

Check if dgt-list of the device is empty. If yes skip further steps.

Using the body = {"commands": ['sh ip access-lists'],

"description": "string",

"deviceUuids": [devid],

"name": "string",

"timeout": 0 }

Call the API <https://ip/api/v1/network-device-poller/cli/read-request> to get task-id.

Using the task-id call the API <https://ip/api/v1/task/task_id> to get file-id.

Give 2 seconds of delay so that DNAC returns proper file-id. Using the file-id call the API <https://ip/api/v1/file/file_id> to get acl content.

The response is as below:

ACLRESP : 'Role-based IP access list Deny IP-00 (downloaded)\\n 10 deny ip\\nRole-based IP access list DenyIP\\n 10 deny ip\\nExtended IP access list IP-Adm-V4-Int-ACL-global\\n 10 permit tcp any any eq www\\n 20 permit tcp any any eq 443\\nRole-based IP access list Permit IP-00 (downloaded)\\n 10 permit ip\\nExtended IP access list VIKRAM\\n 10 permit ip host 31.31.31.4 host 15.1.1.101\\n 20 permit ip host 15.1.1.101 host 31.31.31.4\\nExtended IP access list VIKRAM\_ping\\n 10 permit ip host 31.31.31.4 host 15.1.1.101\\n 20 permit ip host 15.1.1.101 host 31.31.31.4\\nRole-based IP access list alpha\_acl-01 (downloaded)\\n 10 permit udp\\n 20 permit ip\\nExtended IP access list implicit\_deny\\n 10 deny ip any any\\nExtended IP access list implicit\_permit\\n 10 permit ip any any\\nExtended IP access list preauth\_v4\\n 10 permit udp any any eq domain\\n 20 permit tcp any any eq domain\\n 30 permit udp any eq bootps any\\n 40 permit udp any any eq bootpc\\n 50 permit udp any eq bootpc any\\n 60 deny ip any any\\nRole-based IP access list testacl-02 (downloaded)\\n 10 deny icmp\\nExtended IP access list vikram\\n 10 permit udp 10.76.160.0 0.0.31.255 eq snmp 171.68.226.64 0.0.0.63\\nMSMRplusBorder#"},"FAILURE":{},"BLACKLISTED":{}}}]'

ACL-LIST: ['', '', '', '', '', [[['Permit IP-00']], [['Permit IP-00'], ['Deny IP-00'], ['testacl-02', 'alpha\_acl-01', 'Permit IP-00']], [['Deny IP-00'], ['testacl-02'], ['Deny IP-00'], ['Deny IP-00'], ['Deny IP-00'], ['Deny IP-00'], ['Deny IP-00'], ['Deny IP-00'], ['Deny IP-00']]], '']

For each acl in acl-list find for Role-based access list ‘acl’ and parse the corresponding acl content.

Create two Data frames for each network device data as shown below

One data frame contains list of vrfnames, sgt’s and ip address. Another data frame contains list of sgt’s, dgt’s, sgacl and aclcontent

A screenshot of a cell phone

Description automatically generatedA close up of a device

Description automatically generated

deny icmp

testacl

17

0

Global

Compare data frame of ISE with data frame of network device

Part of data frame extracted from ISE data frame for dgt 12

DGT SGT SGACL ACL\_ISE

0 12 0 Permit IP permit ip

Part of data frame extracted from network device for lgt 12

DGT SGT SGACL ACL\_device

0 12 0 Permit IP permit ip

Both data frames are merged on DGT, SGT and SGACL

1.DGT 2.SGT 3.SGACL 4.ACL\_ISE 5.ACL\_Device

0 12 0 Permit IP permit ip permit ip

Check consistency between ISE policies and policies configured on the device

For every entry for SGT in network device data frame compare it with ISE data frame entries and policy download is okay if everything matches.

PDF Generation

Reportlab and numpy is used for pdf generation. Create a simple document template named “table.pdf”. Convert data frames into numpy arrays and create a table from array. Append the required paragraph elements and table elements to a list and build the document. Table of contents page is also generated with internal links to corresponding page numbers. If the SGT, SGACL in ISE doesn’t exist in device or vice-versa ‘nan’ cells are highlighted in red. If the acl content doesn’t match cells are highlighted in yellow.

A screenshot of a cell phone

Description automatically generated

If the DNAC or ISE credentials are wrong display the same in the pdf.

A screenshot of a cell phone

Description automatically generated

If correct credentials are shown display the network device information and policy comparison as shown below.

A screenshot of a cell phone

Description automatically generated