INSRUCTIONS FOR AUTO DISCOVERY AND NETWORK MAPPER FOR ZABBIX

Script developed using Python and Pyzabbix libraries

Abstract

This guide contains guides on how to use the codebase to manage scripts of auto discovery and network mapping when provisioning Zabbix instances on client environments.

Aman Thapa Magar

aman.yhbs@gmail.com

Table of Contents

1 Pre-requisites:

Python, pip and pyzabbix package are required to connect to Zabbix API to perform operations.

```
#!/bin/bash

# Run the first Python script
echo "#################### Running AutoDiscovery Script ##################"
python3 test2.py
echo "Auto Discovery Completed"

echo "############### Restarting Zasya Services ####################"
expect restart-server.exp

echo "Zasya Services Restarted"

#wait for 2 minutes for data to be fetched after restart

echo "################## Waiting for Data to be Fetched for Graphing ##############"

sleep 120

# Run the second Python script
echo "################ Running Auto Network Mapper Script ################"
python3 netmapscript.py
echo "Network Map Created. Please arrange the map to your suitable topology and perform any other manual changes to your liking !!"
```

1. The script is controlled by bash file **main.sh** which contains the auto discovery function **(test2.py)** and network mapping functions **(netmapscript.py & netmapfunc.py)** called. In the main bash file. There are two scripts called in an interval of 2 min (i.e., this is done to let the data be fetched from desired networks and so the network mapper finds these data when fetching via json responses)
2. The **netmapfunc.py** script handles all necessary function to request calls from Zabbix API for desired hostgroup.
3. The **netmapscript.py** handles the linking and combining image in the map space and provides devices appropriate hostname and IP address fetched from the collected data itself.

2 Modifiable Values:

2.1 Autodiscover and Actions:

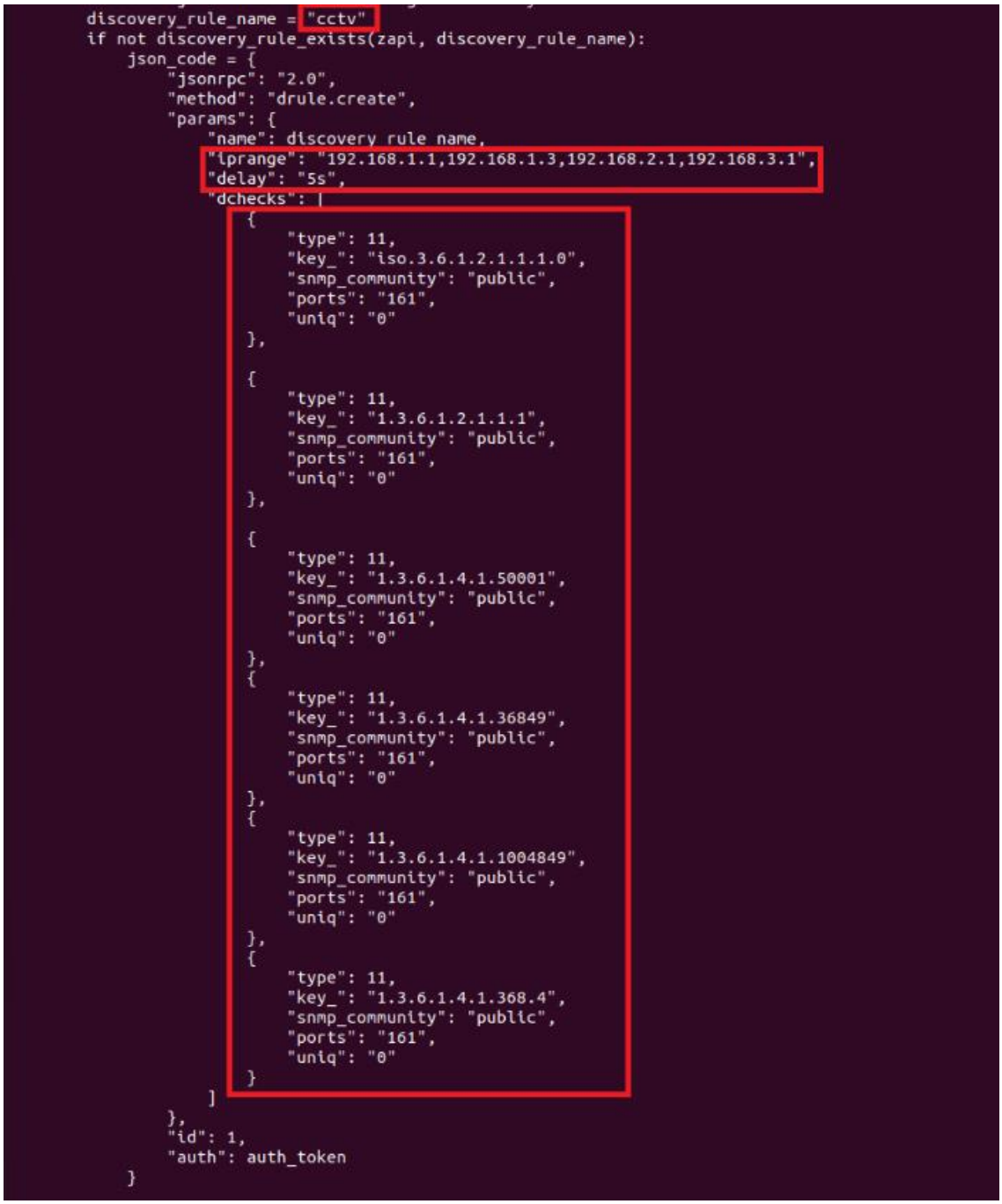
```
# Replace with your Zabbix server information
zabbix_url = 'http://192.168.1.2/api_jsonrpc.php'
zabbix_user = 'Admin'
zabbix_password = 'ZABBIX_ADMIN_PASS'

zapi = ZabbixAPI("http://192.168.1.2")
zapi.login(zabbix_user, zabbix_password)
```

As previous, change the above values of **zabbix_url, Zabbix_user and Zabbix_password** and **zapi (http://x.x.x.x)** to your liking, you maybe confused as to why there are two different urls called. To clear out that notion, both of these statements are required for the script to run.

- Zapi is using pyzabbix function to fetch IDs to create new desired functions
- Zabbix_url is being used to push json code using subprocess directly inside the Zabbix server.

```
    # Create or get the host group named "cctv"
    hostgroup_name = "IP Camera"
    hostgroup_id = create_or_get_hostgroup(zapi, hostgroup_name)
```

A hostgroup name is also created with the name IP Camera, you can change this to your liking.



```
discovery_rule_name = "cctv"
if not discovery_rule_exists(zapi, discovery_rule_name):
    json_code = {
        "jsonrpc": "2.0",
        "method": "drule.create",
        "params": {
            "name": discovery_rule_name,
            "iprange": "192.168.1.1,192.168.1.3,192.168.2.1,192.168.3.1",
            "delay": "5s",
            "dchecks": [
                {
                    "type": 11,
                    "key_": "iso.3.6.1.2.1.1.1.0",
                    "snmp_community": "public",
                    "ports": "161",
                    "uniq": "0"
                },

                {
                    "type": 11,
                    "key_": "1.3.6.1.2.1.1.1",
                    "snmp_community": "public",
                    "ports": "161",
                    "uniq": "0"
                },

                {
                    "type": 11,
                    "key_": "1.3.6.1.4.1.50001",
                    "snmp_community": "public",
                    "ports": "161",
                    "uniq": "0"
                },
                {
                    "type": 11,
                    "key_": "1.3.6.1.4.1.36849",
                    "snmp_community": "public",
                    "ports": "161",
                    "uniq": "0"
                },
                {
                    "type": 11,
                    "key_": "1.3.6.1.4.1.1004849",
                    "snmp_community": "public",
                    "ports": "161",
                    "uniq": "0"
                },
                {
                    "type": 11,
                    "key_": "1.3.6.1.4.1.368.4",
                    "snmp_community": "public",
                    "ports": "161",
                    "uniq": "0"
                }
            ]
        },
        "id": 1,
        "auth": auth_token
    }
```

The above image is for creating a Autodiscover rule with the name **“cctv”** this can be modified to your liking before deploying the script on the server. Likewise, IP address range can be defined within the json params. (i.e. one thing I noticed why your script of Autodiscover was not working was you were using a very large /16 pool to determine the ip address of the devices. The problem of Zabbix or one of drawbacks is it goes by sending hello packets one by one to each ip starting from the range till it reaches the desired Ip range block for example: if it traverses using 192.168.0.0/16 and your desired IP range to be scanned is 192.168.10.0/24 then, it will start scanning and sending one packet to each IP one at a time like 192.168.0.1, 192.168.0.2……etc and so on. Which is a not a reliable source to stay with. Hence, to eradicate

the issue, if the ip addresses range are known that we can eliminate the process of scanning each individual network range and automatically add devices.)

NOTE: what I am talking about can be found using tail -f /var/log/Zabbix/Zabbix_server.log file but you will need to enable the /16 pool and restart the server to see the errors.

The number of blocks that I have added as a tests include the following snmp oid as I saw in your presentation and are of following vendors
mobotix - 1.3.6.1.2.1.1.1

Hikivision - 1.3.6.1.4.1.50001

Hanwha - 1.3.6.1.4.1.36849

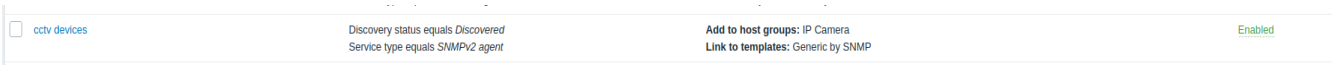
Dahua - 1.3.6.1.4.1.1004849

Axis - 1.3.6.1.4.1.368.4

(I found some, but some don’t have online mib file and should you require necessary adjustment please add a copy a set of block and just define the oid’s you won’t need to change anything.)

```
# Your adjusted code for creating a discovery action
discovery_action_name = "cctv devices"
if not discovery_action_exists(zapi, discovery_action_name):
    json_action = {
        "jsonrpc": "2.0",
        "method": "action.create",
        "params": {
            "name": discovery_action_name,
            "eventsource": 1,
            "filter": {
                "evaltype": 0,
                "conditions": [
                    {
                        "conditiontype": 10,
                        "operator": 0,
                        "value": 2
                    },
                    {
                        "conditiontype": 8,
                        "operator": 0,
                        "value": 11
                    }
                ]
            },
            "operations": [
                {
                    "operationtype": 6,
                    "optemplate": [
                        {
                            "templateid": "10563"
                        }
                    ]
                },

                {
                    "operationtype": 4,
                    "opgroup": [
                        {
                            "groupid": hostgroup_id
                        }
                    ]
                }
            ]
        },
        "id": 1,
        "auth": auth_token
    }
```

After the discovery rule is created it will not then proceed to create the discovery action rule, where pretty much you don't have to really do anything in this part but should you require to change the rules and operations you will need to read the Zabbix API docs especially the one with actions where they have defined set of params for each function. Some are not that precise so you might need to use postman to fetch using **action.get** to see what ID's are associated with what kind of actions and operations for now I believe this will work for all kinds of devices and no changes are needed. The above action will do the following in the Zabbix UI



2.2 Network Mapper:

```
min_coordinate = 300
max_coordinate = 800
zabbixuser = 'Admin'
zabbixpassword = 'ZABBIX_ADMIN_PASS'

trigger1 = 'Generic SNMP: High ICMP ping response time'
trigger2 = 'Generic SNMP: High ICMP ping loss'
trigger3 = 'Generic SNMP: Unavailable by ICMP ping'
trigger1color = 'FF6F00'
trigger2color = 'FF6F00'
trigger3color = 'DD0000'
custom_icon_path = 'icon/cctv_(64).png'

zapi = ZabbixAPI("http://192.168.1.2")
zapi.login(zabbixuser, zabbixpassword)
#print('Enter the hostgroup for which you want to make a map, format, exact synt
ax needed')
my_hostgroup = 'IP Camera'
```

The values above are modifiable to an extent in netmapscript.py such as it determines which hostgroup to use when creating the maps. As the network maps are being created based on the hostgroup ID, for my testing purposes I have just directly put values as **'IP Camera'**. If you want to add the hostgroup name yourself then you can just uncomment the print and add a line **my_hostgroup = input('')** and comment out the **my_hostgroup = 'IP Camera'**. Triggers are set of events that define the severity of any problems within the device based on vendor base templates, in this section I am using **Generic SNMP: xxxxxxx Descriptions** because different vendors have different templates, since we are using to hostgroup to map network for the hosts present inside of it. We should understand the fact that not every device included inside the hostgroup will use the same template as it is not feasible, instead we should use a common monitoring template so we can accommodate the basic requirements for the map to discover the host and display on the webpage. The main goal being eradicating the need to clone devices and add links in the map. Custom modifications can be done after the generation of the maps. The colors define the set of hex codes based on severity level. Icon has been added to the file to emulate cctv image, if it exists already then it will skip uploading the image otherwise it inserts an image and then uses the icon inside the map. Likewise, url, username and password can be modified based on your requirements.

3 Results

Automatic Creation of Hostgroup, Autodiscover Rules, Action and Network Map so this file can be called after installation script by adding a small bash line of **main.sh**