**Introduction**

Vectra Automated Response

This is a framework meant to allow for easy integration of any third-party security vendor.

Based on various input parameters (details below), the script returns a list of Host/Accounts/Detections to be blocked/unblocked.

The code defines an [abstract class](https://sourcecode.vectra.io/projects/CS/repos/var/browse/third_party_clients/third_party_interface.py) which third-party clients must extend, which allows for easy integration with the workflow implemented by the base script.

Since adding a new third party integration only requires to extend that class, I'd encourage to use this framework for any new integrations being built.

# Third party integrations

Currently, the following third party integrations are implemented:

1. Bitdefender
2. Call an external program
3. Cisco AMP
4. Cisco FMC
5. Cisco ISE
6. Cisco Meraki
7. Cisco PxGrid
8. ClearPass
9. Endgame
10. Fortinet Firewalls (FortiOS)
11. Harmony
12. McAfee EPO
13. Palo Alto Network Firewalls (Panorama or not)
14. PAN Cortex
15. Pulse Secure NAC
16. Sophos Firewall
17. Static destination IP blocking
18. Trendmicro ApexOne
19. Trendmicro CloudOne
20. Trendmicro VisionOne
21. VMWare vSphere
22. WatchGuard
23. Windows (direct PowerShell commands to shutdown host)
24. WithSecure Elements

Integration-specific documentation can be found in the [relevant folders](https://sourcecode.vectra.io/projects/CS/repos/var/browse/third_party_clients/) of the third party integrations.

# Requirements

Install the python module requirements utilizing pip3:

pip3 install -r requirements.txt

Vectra API Tools version 2.4+ is required to support account based groups.

# Workflow

The script supports host, account, and detection based blocking. Parameters defining what host/account/detections get blocked are defined in the [config.py](https://sourcecode.vectra.io/projects/CS/repos/var/browse/config.py) file.

# Secrets Storage

This script utilize the Python Keyring package to securely store secrets locally to where the script is ran. The script will check the default keyring for the necessary secrets and prompt the user if they are not found. The default configuration is to store the secrets after input. If user desires not to store the secrets, utilize the **--no\_store\_secrets** option.

# Getting a Vectra API token for Vectra Detect API (v2).

Vectra Detect API v2 utilizes token based authentication. To create a token, login into Vectra, go to "My Profile" and click to create an API token. Note that only local accounts can generate API tokens.

Vectra API tokens will be linked to the user that created them, and inherit the rights of that user. Any actions done using that API token will also show under the same username in the audit logs.

You may want to create a separate user for the API integration for audit purposes, and only give it fine-grained RBAC rights. For the integration to work, the user will need:

* Read access to Hosts
* Read access to Accounts
* Read access to Detections
* Read access to "Manage - Groups"
* Read/Write access to tags
* Read/Write access to Notes & Other User's Notes

# Getting a Vectra API Client ID and Secret Key for Vectra Platform API (v3).

Access to the Vectra Platform API is done through the creation of an API Client. Creation of an API Client will provide a set of OAuth 2.0 credentials that will be used to gain authorization to the Vectra Platform API. Please note that management of API Clients is restricted to Detect users with the role of “Super Admin”. To create an API client, log into your Detect portal and navigate to Manage > API Clients. From the API Clients page, select ‘Add API Client’ to create a new client. Once created, be sure to record your Client ID and Secret Key for safekeeping. You will need these two pieces of information to obtain an access token from the Vectra Platform API. An access token is required to make requests to all of the Vectra Platform API endpoints.

Vectra API Clients will inherit the rights of the role selected during creation.

You may want to create a separate role for the API integration for audit purposes, and only give it fine-grained RBAC rights. For the integration to work, the API Client will need:

* Read access to Hosts
* Read access to Accounts
* Read access to Detections
* Read access to "Manage - Groups"
* Read/Write access to tags
* Read/Write access to Notes & Other User's Notes

## Host-based blocking

The goal of host-based blocking is identifying internal hosts who need to be prevented from being able to further communicate internally and/or externally. The blocking will happen on host specific attributes, such as for instance the internal IP address, the MAC address or the hostname.

There are multiple parameters within the [config.py](https://sourcecode.vectra.io/projects/CS/repos/var/browse/config.py) file which define how hosts are being selected for blocking:

1. BLOCK\_HOST\_TAG: defines a tag that when set on a host will cause that host to be blocked.
2. NO\_BLOCK\_HOST\_GROUP\_NAME: defines a group name, where all members of that group will not be blocked. That group needs to be created manually on the Detect UI, it will not be created by the script.
3. BLOCK\_HOST\_GROUP\_NAME: defines a group name, where all members of that group will be blocked. That group need to be created manually on the Cognito UI, it will not be created by the script.
4. BLOCK\_HOST\_THREAT\_CERTAINTY: defines a threat and certainty score threshold, above which host will get blocked. The middle variable can be either *\_and\_* or *\_or\_*, defining how the threat and certainty conditions are threated.
5. BLOCK\_HOST\_URGENCY: defines an urgency score threshold, above which host will get blocked. Only used with V3 API Can't have both BLOCK\_HOST\_THREAT\_CERTAINTY and BLOCK\_HOST\_URGENCY. If both provided and V3 is True, BLOCK\_HOST\_URGENCY will be used. To use BLOCK\_HOST\_THREAT\_CERTAINTY set BLOCK\_HOST\_URGENCY = None
6. BLOCK\_HOST\_DETECTION\_TYPES: this is a list containing specific detection types, which when present on a host will cause that host to be blocked.
7. BLOCK\_HOST\_DETECTION\_TYPES\_MIN\_TC\_SCORE: minimum Threat and/or Certainty scores for an HOST with specified detection types before it will be blocked.

Besides this, the NO\_BLOCK\_HOST\_GROUP\_NAME Defines a group name from which all members will never be blocked. Users need to create that group themselves, it is not created automatically by the script.

**Important:** when blocking conditions are no longer fulfilled for a host, either because the blocking tag was removed, its score decreased, group membership was revoked, or the specific detection types causing blocking were fixed, the host will be automatically unblocked by the script on the next run.

## Account-based blocking

The goal of account-based blocking is to disable or otherwise restrict an account to limit its usage by an adversary. Currently, the only module supporting this functionality is by calling an external program with the external\_call module.

1. BLOCK\_ACCOUNT\_TAG: defines the tag applied to an account which results in the block\_account methods being called for the configured clients
2. NO\_BLOCK\_ACCOUNT\_GROUP\_NAME: defines a group name, where all members of that group will not be blocked. That group needs to be created manually on the Detect UI, it will not be created by the script.
3. BLOCK\_ACCOUNT\_GROUP\_NAME: defines a group name, where all members of that group will be blocked. That group need to be created manually on the Detect UI, it will not be created by the script.
4. BLOCK\_ACCOUNT\_THREAT\_CERTAINTY: defines a threat and certainty score threshold, above which host will get blocked. The middle variable can be either *\_and\_* or *\_or\_*, defining how the threat and certainty conditions are treated.
5. BLOCK\_ACCOUNT\_URGENCY: defines an urgency score threshold, above which host will get blocked. Only used with V3 API Can't have both BLOCK\_ACCOUNT\_THREAT\_CERTAINTY and BLOCK\_ACCOUNT\_URGENCY. If both provided and V3 is True, BLOCK\_ACCOUNT\_URGENCY will be used. To use BLOCK\_ACCOUNT\_THREAT\_CERTAINTY set BLOCK\_ACCOUNT\_URGENCY = None
6. BLOCK\_ACCOUNT\_DETECTION\_TYPES: this is a list containing specific detection types, which will always result in the account being blocked.
7. BLOCK\_ACCOUNT\_DETECTION\_TYPES\_MIN\_TC\_SCORE: minimum Threat and/or Certainty scores for an account with specified detection types before it will be blocked.

## Detection-based blocking

The goal of detection-based blocking is identifying detection containing external components (IP or domain), which can then be blacklisted on various security tools to prevent communication towards those from any internal machine.

Most internal-focused third party clients, such as NACs or endpoints will not implement detection-based blocking, as they are not able to block public IPs/domains. This is mainly relevant for Firewall specific third party clients. Nevertheless, using detection based blocking with a mix of clients supporting it and not is not an issue, but will cause warnings to be logged when executing the script.

Since this usually will block the IP/domain for the whole environment, **extreme care is advised**, as in the case of false-positives it can have a large impact on the network.

There are multiple parameters within the [config.py](https://sourcecode.vectra.io/projects/CS/repos/var/browse/config.py) file which define how detections are being selected for blocking:

1. EXTERNAL\_BLOCK\_HOST\_TC: defines host threat/certainty score above which all detections present on that host will get marked for blocking (or at least any external component present in those detections).
2. EXTERNAL\_BLOCK\_DETECTION\_TAG: defines a tag that when set on a detection will cause all external components of that detection to be blocked.
3. EXTERNAL\_BLOCK\_DETECTION\_TYPES: this is a list containing specific detection types, which will always have their external components automatically blocked. Any valid detection type will be accepted by the script, but it only makes sense for detections with an external component, thus Botnet, Command&Control or Exfil detections.

# Configuring the Third-Party clients used

Users need to configure which third-party clients they intend the script to use. This configuration needs to be done directly in the [config.py](https://sourcecode.vectra.io/projects/CS/repos/var/browse/config.py) file. The user can manually update the [config.py](https://sourcecode.vectra.io/projects/CS/repos/var/browse/config.py) and the [third\_party\_clients/third\_party\_client/third\_party\_client\_config.py], or utilize the [var\_config\_helper.py](https://sourcecode.vectra.io/projects/CS/repos/var/browse/var_config_helper.py) script. To run the script:

python3 var\_config\_helper.py

The script will pull all of the pertinent variables and present them to the user for configuration. When the script is complete, all configurations will be written to the appropriate configuration file(s).

## Instantiating the clients

By default, all clients configured in the [config.py](https://sourcecode.vectra.io/projects/CS/repos/var/browse/config.py) file are automatically instantiated.

## Passing the third party client instances to the script

Once all required third party clients have been instantiated, they are appended to the list argument \_(third\_party\_client)\_. When the instantiation call of the VectraAutomatedResponse() class is called, all required third party clients will be available to that class.

## Selecting what block types to run

Users can also configure if they want to run only host-based, account-based, detection-based blocking or all.

If one type is not desired, you can comment out the corresponding code blocks:

# Those 3 lines handle host-based blocking; comment them out if you don't want it

hosts\_to\_block, hosts\_to\_unblock = var.get\_hosts\_to\_block\_unblock()

var.block\_hosts(hosts\_to\_block)

var.unblock\_hosts(hosts\_to\_unblock)

# Those 3 lines handle detection-based blocking; comment them out if you don't want it

detections\_to\_block, detections\_to\_unblock = var.get\_detections\_to\_block\_unblock()

var.block\_detections(detections\_to\_block)

var.unblock\_detections(detections\_to\_unblock)

### Using the external\_call module

The external call module's configuration file contains a list per blocking/unblocking action which supplies the command and required arguments to the external program to execute the desired functionality.

HOST\_BLOCK\_CMD = []

HOST\_UNBLOCK\_CMD = []

ACCOUNT\_BLOCK\_CMD = []

ACCOUNT\_UNBLOCK\_CMD = []

DETECTION\_BLOCK\_CMD = []

DETECTION\_UNBLOCK\_CMD = []

To configure commands, each element of the command must become an element in the appropriate list. As an example, if the desired command to execute is to ping source IP 5 times, the list would be configured as such:

HOST\_BLOCK\_CMD = ['ping', '-t', '5']

The default code in the external\_call.py module is written to supply the command and arguments followed by the host IP or account name, by appending the host IP or account name to the command list.

def block\_host(self, host):

if HOST\_BLOCK\_CMD:

id = uuid.uuid4()

cmd = HOST\_BLOCK\_CMD

cmd.append(host.ip)

r = subprocess.run(cmd)

The IP or account name may need to be inserted into command list vs appending, this requires modification to the default code.

## Static Destination IP Blocking

Blocking of static IPs/Subnets is supported with FortiGate and Palo Alto firewalls by specifying IPs/subnets one per line, in the main configuration file config.py. The parameter and default file name are:

STATIC\_BLOCK\_DESTINATION\_IPS = 'static\_dst\_ips\_to\_block.txt'

IPs or subnets can be added to the file, one per line, for blocking as a destination address. Example:

1.1.1.1

2.2.2.0/24

3.3.3.3

Removing an entry(ies) from the configuration file will result in the IP/subnet being removed from the firewall during the next run of the script.

## Running the script

The script can be run manually, via a cron job, or as a service. If running as a service, specify the --loop flag to run the script in a continuous loop with the pause time configured in the config.py file's variable SLEEP\_MINUTES.

### Additional options

#### Monitoring for IP changes

Modules may support attempting to re-block a host (re-grooming) if that host's IP has changed since it was originally blocked. To enable re-grooming for supported modules specify the --groom flag.