StrangeHiveDocs

Release 0.1

Pcktech

Apr 23, 2022

CONTENTS

TheHive is an Incident Response Platform enabling documentation and coordination of identification and response to cyber threats.

Official Sites include:

- TheHive Project
- TheHive 4 Github
- StrangeBee TheHive 5+

Note: This documentation is under active development. A-Z Setup Guide for TheHive 4 exists as a Word doc, but need to sanitize and convert.

CHAPTER

ONE

CONTENTS

1.1 TheHiveHooks

1.1.1 Requirements

The following instructions are based on CentOS7 and Oracle Enterprise Linux 8, and assumes TheHiveHooks is installed on the same server as TheHive.

- TheHive 4.x
- Python 3.6
- TheHive4Py 1.8.1+
- Command Line/Terminal access to TheHive server
- Root/Sudo access at the Command Line/Terminal on TheHive server
- Internet/Web access from TheHive Server (or local copies of all packages reference for localinstall)
- TheHive 4.x org-admin role/profile for desired organizations

1.1.2 Installation

Install Supporting Packages

1. Elevate to Root (for convenience): sudo su -
2. If you can only access internet sites via a web proxy:
    a. set HTTP_PROXY=http://**IPADDRESS**:**PORT**
    b. set HTTPS_PROXY=http://**IPADDRESS**:**PORT**
    c. set PIP_CERT=/etc.pki/tls/certs/ca-bundle.crt (assumes your updated the OS Trusted Certificates with the Certificate Authority used by the web proxy)
3. Ensure PIP is up to date: python3 -m pip install --upgrade pip
4. Install Supervirod: python3 -m pip install supervisor
5. Install Virtual Env for Python: yum install virtualenv
6. Install Git for TheHiveHooks download: yum install git
7. If you use a web proxy, set Git's proxy:
    a. Set: git config --global http.proxy http://**IPADDRESS**:**PORT**

b. Verify: git config --global --get-regex http.*

Install TheHiveHooks

8. Log in to TheHive's Web UI as an org-admin for the desired organization (or to the 'admin' organization):
   - IMPORTANT: If you have more than one organization you wish to use TheHiveHooks with, you must perform the following for each desired organization **and** the username for each organization's TheHiveHooks account **must** be unique. If you use the exactly same username it will only work for the first organization where that account was created (its default organization).
   a. Create a local account to be used by TheHiveHooks (e.g. username: thehivehooks@thehive.local) in the desired organization with the org-admin role.
   b. Click the Create API Key button for this local account, then click the Reveal API Key - note this somewhere safe for later.
9. Return to the command line/terminal, and change directories to /opt: cd /opt
10. Download TheHiveHooks: git clone https://github.com/TheHive-Project/TheHiveHooks.git
11. Verify /opt/TheHiveHooks was created: ls /opt
12. Change directories: cd /opt/TheHiveHooks
13. Create a virtual Python instance for TheHiveHooks (so any updates made to the global/OS instance don't break any older and required versions that might be needed later for TheHiveHooks or TheHive4Py):

a. Start the creation (can take 1-5 minutes): virtualenv venv
   - Troubleshooting: If creation appears stuck at "Installing setuptools, pip, wheel", try clearing the user profile's cache folder in a different terminal session (or CTRL+C to cancel, clear, and relaunch the attempt):
     – Verify the cache folder exists: ls -l ~/.cache/pip
     – Clear the cache folder: rm -rf ~/.cache/pip
     – Verify the cache folder no longer exists: ls -l ~/.cache

b. Verify it was created: ls venv (AKA ls /opt/TheHiveHooks/venv)

14. Enter the virtual environment: source /opt/TheHiveHooks/venv/bin/activate
15. If you use a web proxy, set the session's environmental variables (HTTP_PROXY, HTTPS_PROXY, PIP_CERT).
16. Verify PIP is up to date: python3.6 -m pip install --upgrade pip
17. Install TheHive4Py: python3.6 -m pip install thehive4py
18. Install TheHiveHooks' required modules: python3.6 -m pip install -r requirements.txt (AKA python3.6 -m pip install -r /opt/TheHiveHooks/requirements.txt)
19. Install Gunicorn: python3.6 -m pip install gunicorn
    - By default TheHiveHooks will use Flask, which is sufficient for dev/test, but not recommended for production. Gunicorn will fulfill Flask's role later.
20. Install Keyring: python3.6 -m pip install keyring
21. Install Keyring's crypted file storage: python3.6 -m pip install keyrings.cryptfile
    - Keyring and Keyring Cryptfile will be used to store the API Keys in a slightly more secure method than hard-coding them in TheHiveHooks script later. It separates the actual API Key from the script so you can backup the script without also inadvertantly storing your API Keys in plain text in a less secure location (or if you upload

them to github for whatever reason). The true security of your API Keys will depend on how the OS is configured -- anyone able to access the filesystem can find the password to decode the API Keys.

- There are more secure methods of doing this. I gave it a try, but it didn't work or required too many packages and too much time to untangle so if this is a concern look into secure ways of storing script credentials.

22. Open a Python session to configure Keyring: python

a. Import: from keyrings.cryptfile.cryptfile import CryptFileKeyring

b. Object: kr = CryptFileKeyring()

c. Add API Key: kr.set_password("**ACCTNAME**", "API", "**APIKEYHERE**")

- The first time you add a password to keyring it will prompt you to create a password for the keyring. This password will be used in TheHiveHook script later to enable it to retrieve/use the desired API Key.
- Use the 'Add API Key' syntax above as many times as desired to add as many API Keys as desired to the keyring. Make sure the ACCTNAME is unique each time.

d. To verify a password was added: kr.get_password("**ACCTNAME**", "API")

e. Once all passwords are added, press CTRL+D to exit the python session.

23. Exit the Virtual Environment: deactivate

Configure TheHiveHooks

24. Edit the launcher: vi /opt/TheHiveHooks/thehive_hooks/__init__.py

```
from flask import Flask
import logging
from logging.handlers import RotatingFileHandler
import os
from pyee import EventEmitter

# Declare the logger
app_dir = os.path.join(os.path.dirname(os.path.abspath(__file__)), '..')
pathLog = app_dir + '/thehive-hooks.log'
handler = RotatingFileHandler(pathLog, maxBytes=10000000, backupCount=1)
handler.setLevel(logging.INFO)
handler.setFormatter(logging.Formatter(
        '[%(asctime)s] %(levelname)s in %(module)s: %(message)s'
))

# Initialize the app
app = Flask(__name__)
# Gunicorn
if __name__ == '__main__':
        # // App called directly - flask or CLI
        app.logger.addHandler(handler)
else:
        # // App called by gunicorn
        gunicorn_logger = logging.getLogger('gunicorn.error')
        app.logger.handlers = gunicorn_logger.handlers
```

```
        # // Use logging level specified by gunicorn conf
        app.logger.setLevel(gunicorn_logger.level)

app.url_map.strict_slashes = False

## Declare the event emitter
ee = EventEmitter()

import thehive_hooks.controllers
import thehive_hooks.handlers
```

25. Edit the script: vi /opt/TheHiveHooks/thehive_hooks/handlers.py

```
import json
from thehive_hooks import app
from thehive_hooks import ee

from thehive4py.api import TheHiveApi
from thehive4py.models import *
from thehive4py.query import *

# // For URL or API Calls
import urllib
# // RegEx for TaskLog
import re
# // ElasticSearch Action-Reaction requires 1 second pause between them.
from time import sleep
# // For Epoch to Human Time Conversion
from datetime import datetime
# // Email Sending
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from email.header import Header
from email.utils import formataddr

# // Key Store
from keyrings.cryptfile.cryptfile import CryptFileKeyring
kr = CryptFileKeyring()
kr.keyring_key = "KEYRINGPASSWORD"
# See def: hive4ApiByOrg

# ################################################
# // WEBHOOK EE-FN CONNECTOR
def make_handler_func(event_name):
        @ee.on(event_name)
        def _handler(event):
```

```
            app.logger.info('Handle {}: Event={}'.format(event_name, json.dumps(event, indent=4,
            sort_keys=True)))

        return _handler

# // WEBHOOK SUPPORTED EVENTS
events = [
        'AlertCreation',
        'AlertUpdate',
        'CaseArtifactCreation', #Hive3
        'CaseArtifactCreate', #Hive4
        'CaseArtifactJobCreation',
        'CaseArtifactJobUpdate',
        'CaseArtifactJobUpdate',
        'CaseArtifactUpdate',
        'CaseCreation', #Hive3
        'CaseCreate', #Hive4
        'CaseTaskCreation', #Hive3
        'CaseTaskCreate', #Hive4
        'CaseTaskLogCreation', #Hive3
        'CaseTaskLogCreate', #Hive4
        'CaseTaskLogUpdate',
        'CaseTaskUpdate',
        'CaseUpdate',
        'CaseDelete'
]

# // WEBHOOK DEFINE SUPPORTED EE.ON FN
for e in events:
        make_handler_func(e)

# ################################################
# // GENERAL FUNCTIONS
def hive4ApiByOrg(event_organisation):
        global api
        global thehive_apikey

        app.logger.info('Case Org: {}'.format(event_organisation))

        if event_organisation == 'ORG1NAME':
                thehive_apikey = kr.get_password("ORG1APIUSERNAME", "API")
                app.logger.info('Case API Key: ORG1NAME Used: {}'.format(thehive_apikey[0:5]))
        elif event_organisation == 'ORG2NAME':
                thehive_apikey = kr.get_password("ORG2APIUSERNAME", "API")
                app.logger.info('Case API Key: ORG2NAME Used: {}'.format(thehive_apikey[0:5]))
        elif event_organisation == 'ORG3NAME':
                thehive_apikey = kr.get_password("ORG3APIUSERNAME", "API")
```

```
            app.logger.info('Case API Key: ORG3NAME Used: {}'.format(thehive_apikey[0:5]))
        elif event_organisation == 'ORG4NAME':
            thehive_apikey = kr.get_password("ORG4APIUSERNAME", "API")
            app.logger.info('Case API Key: ORG4NAME Used: {}'.format(thehive_apikey[0:5]))
        else:
            app.logger.info('Case API Key: Unknown for Org {}'.format(event_organisation))

        api = TheHiveApi(url='http://127.0.0.1:9000', principal=thehive_apikey, version='4')

# FREE EXAMPLE OF A FUNCTION YOU COULD USE WITH THEHIVEHOOKS TO AUTOMATE THINGS IN THEHIVE -- like ensuring when people add a Task Log that single line breaks appear like single line breaks. It can mess with Code Blocks and Tables, however.
def taskDoubleBreak(event):
    if 'message' in event['details'] and '\n' in event['details']['message']:
        # // Prevent infinite loops by filtering out Webhook-made Updates
        if not 'updatedBy' in event['object'] or ('updatedBy' in event['object'] and not
        str(event['object']['updatedBy']).endswith("thehive.local")):
            log_id = event['objectId']
            original_msg = event['details']['message'] + '\n\n###### _Automatically Converted Single to
            Double Line Break_'

            # // Check if an Adjustment is needed to avoid unnecessary Updates
            regexPattern = re.compile('(?<!\n)\n(?!\n)')
            regexResults = regexPattern.findall(original_msg)
            app.logger.info('Single Breaks Found: {}'.format(len(regexResults)))

            if len(regexResults) > 0:
                app.logger.info('Task Log Line Break Adjustment')
                adjusted_msg = re.sub(r'(?<!\n)\n(?!\n)','\n\n', original_msg)

                ctask_url = "http://127.0.0.1:9000/api/case/task/log/" + log_id
                ctask_criteria = {"message":adjusted_msg}
                ctask_json = json.dumps(ctask_criteria).encode('utf8')
                ctask_auth = 'Bearer ' + thehive_apikey
                ctask_request = urllib.request.Request(ctask_url, data=ctask_json,
                headers={'Authorization':ctask_auth,'Content-Type':'application/json'},
                method='PATCH')
                urllib.request.urlopen(ctask_request)

# ################################################
# TRIGGERED EVENTS THAT CALL FUNCTIONS OR PERFORM ACTIONS
@ee.on('CaseTaskLogCreate')
#TheHive 4 Version
def taskLogCreationHook(event):
    # Dynamically changes the global 'api' variable value to include the correct API Key based on the
    Organization name.
    hive4ApiByOrg(event['organisation'])
```

```
    case_guid = event['rootId']

    if 'message' in event['details'] and '\n' in event['details']['message']:
        taskDoubleBreak(event)
```

Starting TheHiveHooks

26. Create TheHiveHooks' log folder: mkdir /var/log/thehivehooks
27. Configure Log Rotation:

a. Create configuration: vi /etc/logrotate.d/rotatehooklogs.conf

b. Copy-Paste configuration into file:

```
/var/log/thehivehooks/thehivehooks.access.log /var/log/thehivehooks/thehivehooks.err.log
/var/log/thehivehooks/thehivehooks.out.log {
    size 1M
    dateext
    rotate 10
}
```

28. Configure Supervisord to Startup:

a. Generate the default supervisord configuration: echo_supervisord_conf > /etc/supervisord.conf

b. Create the Includes folder: mkdir /etc/supervisord.d

c. Enable Includes (external conf files): vi /etc/supervisord.conf

i. Find the [include] section (usually at the very bottom) and remove the semicolor in front of it and the following "files =" line.

ii. Change the "files =" value from the default to: /etc/supervisord.d/*.conf

d. Create the startup script: vi /etc/rc.d/init.d/supervisord

```
#!/bin/sh
#
# /etc/rc.d/init.d/supervisord
#
# Supervisor is a client/server system that
# allows its users to monitor and control a
# number of processes on UNIX-like operating
# systems.
#
# chkconfig: - 64 36
# description: Supervisor Server
# processname: supervisord

# Source init functions
. /etc/rc.d/init.d/functions
```

```
prog="supervisord"

prefix="/usr/local"
exec_prefix="${prefix}"
prog_bin="${exec_prefix}/bin/supervisord"
PIDFILE="/var/run/$prog.pid"

start()
{
        echo -n $"Starting $prog: "
        daemon $prog_bin --pidfile $PIDFILE
        [ -f $PIDFILE ] && success $"$prog startup" || failure $"$prog startup"
        echo
}

stop()
{
        echo -n $"Shutting down $prog: "
        [ -f $PIDFILE ] && killproc $prog || success $"$prog shutdown"
        echo
}

case "$1" in

        start)
                start
        ;;

        stop)
                stop
        ;;

        status)
                status $prog
        ;;

        restart)
                stop
                start
        ;;

        *)
                echo "Usage: $0 {start|stop|restart|status}"
        ;;

esac
```

e. Allow the startup script to execute (function): chmod +x /etc/rc.d/init.d/supervisord

f. Add the startup script to startup: chkconfig --add supervisord

g. Enable startup script execution: chkconfig supervisord on

h. Apply changes: systemctl daemon-reload

29. Configure Supervisord to run TheHiveHooks:

a. Create TheHiveHooks configuration for Supervisord: vi /etc/supervisord.d/superdhook.conf

```
[program:thehivehooks]
directory=/opt/TheHiveHooks
command=/opt/TheHiveHooks/venv/bin/gunicorn thehive_hooks:app -b 127.0.0.1:5001 --workers 8
--max-requests 1000 --error-logfile /var/log/thehivehooks/thehivehooks.err.log --access-logfile
/var/log/thehivehooks/thehivehooks.access.log --log-level info --log-file
/var/log/thehivehooks/thehivehooks.out.log --capture-output
autostart=true
autorestart=true
#redirect_stderr=True
stderr_logfile=/var/log/thehivehooks/thehivehooks.err.log
stdout_logfile=/var/log/thehivehooks/thehivehooks.out.log
```

30. Start Supervisord: service supervisord start

31. Check Supervisord's status: supervisorctl status

- If 'thehivehooks' shows 'STARTING' for more than 3 seconds there's most likely a problem with TheHiveHooks' __init__.py or handlers.py, or with the Supervisord conf files. Check the applicable log files for error messages.
- Anytime you modify your handlers.py code, you must restart supervisord to apply the code changes: supervisorctl restart -- and you should **always** verify it doesn't get stuck 'STARTING' because of a typo in your code change.

Enabling TheHiveHooks in TheHive

32. Edit TheHive's configuration to call on Gunicorn's Port (which is running TheHiveHooks): vi /etc/thehive/application.conf

a. Scroll to the bottom and make sure the following block exists, or add it:

```
notification.webhook.endpoints = [
        {
                name: local
                url: "http://127.0.0.1:5001/webhook"
                version: 0
                auth: {type: "none"}
                includedTheHiveOrganisations: ["*"]
                excludedTheHiveOrganisations: []
        }
]
```

33. Restart TheHive: systemctl restart thehive

34. Verify TheHive Service remains running for more than 30 Seconds: systemctl status thehive
    - If the service fails to start, verify the application.conf modification didn't remove anything or if there's a typo in any new configuration added.
35. For **each** organization where you want TheHiveHooks activated/functioning, you must perform the following steps to finish enabling TheHiveHooks using the API key of the org-admin account (with a username unique to that organization) that was created earlier:

a. Securely enter the API Key so it isn't plastered all over your command history: read -s -p 'Enter the API Key: ' thehive_apikey

b. Use the following command to instruct TheHive to use TheHiveHooks for the organization of the account used to run the command:

```
curl -XPUT -H 'Content-type: application/json' -H "Authorization: Bearer $thehive_apikey"
http://127.0.0.1:9000/api/config/organisation/notification -d '
{
        "value": [
                {
                        "delegate": false,
                        "trigger": { "name": "AnyEvent"},
                        "notifier": { "name": "webhook", "endpoint": "local" }
                }
        ]
}'
```

c. Verify the webhook was activated successfully: curl -H 'Content-type: application/json' -H "Authorization: Bearer $thehive_apikey" http://127.0.0.1:9000/api/config/organisation/notification

- You should receive output that looks like the following:

{"path":"notification","defaultValue":[],"value":[{"delegate":false,"trigger":{"name":"AnyEvent"},"notifier":{"name":"

36. Exit the Sudo Su Elevation: logout
37. Test that your TheHiveHook code works as expected in TheHive.