Heist Documentation

None

Jun 28, 2023

CONTENTS

CHAPTER

ONE

HEIST

Heist creates network tunnels for distributing and managing agents. While it has been originally built to deploy and manage Salt minions, it can be used to distribute and manage other agents or plugins if extended to do so.

Using Heist is very easy. Start by downloading Heist. Just install via `pip`:

```
pip install heist
```

1.1 Heist Manager

In order to run Heist you will need to specify a Heist manager. The default Heist manager included with Heist is called `test`. When you run Heist with the `test` manager it will just log a statement stating Heist is installed correctly and to use a fully functioning Heist manager. This is used just for testing your setup in Heist.

Another example of a Heist manager is `salt.minion` which is used to managed a Salt minion. It is made available by installing heist-salt using `pip`:

```
pip install heist-salt
```

1.2 Making Your Roster

A Roster is a file used by Heist to map login information to the systems in your environment. This file can be very simple and just needs to tell Heist where your systems are and how to log into them via SSH. Open a file called *roster.cfg* and add the data needed to connect to a remote system via SSH:

```
system_name:
  host: 192.168.4.4
  username: fred
  password: freds_password
```

Note: This example is very simple, Heist supports virtually all available authentication options for SSH.

All roster files typically live inside of a roster directory. But to get started we will execute a single roster file with Heist using the `heist-salt` manager:

```
heist salt.minion -R roster.cfg
```

Assuming your roster is correct, Heist will now connect to the remote system, and deploy the salt minion binary.

1.3 Tear Down

Heist is able to automatically clean up as well! Just soft kill your Heist application and it will reach out to all connections, tell them to remove the deployed artifacts from the target systems and stop the service! Like a proper Heist there should be no evidence left behind!

1.4 Using Heist to Deploy Salt Minions

If you want to use Heist to deploy and manage Salt, you will need to install heist-salt.

1.5 Using the Test manager in Heist

You can use the `test` manager in Heist to ensure your Heist install is setup correctly. You just need to run:

```
heist test -R roster.cfg
```

If successful, the test manager will log that you successfully used the `test` manager and inform you that you need to use a different manager for full functionality. This test manager will ensure functionality of Heist up until it calls a manager. It will detect roster rendering failures and dependency install issues.

CHAPTER

TWO

ROSTERS

Rosters is the system that is used to define the target systems to create connections to with Heist. The default roster system is called *flat* and uses the POP `rend` system to render the datasets.

Note: By using the `flat` roster you can make roster files using yaml, json, toml etc. and template the files making it easy to allow for logic to make larger lists easier. Don't worry! You don't need to know anything about `rend` to use rosters. Just know that there is a robust system under the hood to make your life easier!

Defining a basic roster is easy:

```
192.168.0.24:
  username: harry
  password: foobar
```

In this roster we are using the default yaml `rend` system. It is also very simple because it is just a password login. Heist supports logging into systems using virtually any login mechanism available to SSH. The options are mapped directly to asyncssh and can be found here: https://asyncssh.readthedocs.io/en/latest/api.html#asyncssh.SSHClientConnectionOptions

You can change the `rend` system used to render the rosters by setting the `renderer` option. By default this is set to yaml.

2.1 Common Roster Options

Use password authentication:

```
192.168.0.24:
  username: harry
  password: foobar
```

Using an SSH key for authentication:

```
192.168.0.24:
  username: harry
  client_keys:
    - /path/to/ssh/key
```

Using an SSH key with a passphrase for authentication:

```
192.168.0.24:
  username: harry
  client_keys:
    - /path/to/ssh/key
  passphrase: "password"
```

Using sudo:

```
192.168.0.24:
  username: harry
  password: foobar
  sudo: True
```

Using sudo with a tty. You need to set *tty* to True if you are using sudo and a password. Heist will interactively input the password defined in the roster when sudo asks for a password. If you have NOPASSWD set in /etc/sudoers for the defined user you do not need to set *tty* to True unless requiretty is set on your target. Here is an example:

```
192.168.0.24:
  username: harry
  password: foobar
  sudo: True
  tty: True
```

Using sudo with a tty while defining the term_type and term_size. The *term_type* and *term_size* are defining Async-SSH's *term_type* and *term_size* options. The *term_type* defines the terminal type to use for the session. By default this is *xterm-color*. The *term_size* defines the terminal width and height in characters. By default *term_size* is set to (80,24). Here is an example of setting both *term_type* and *term_size*:

```
192.168.0.24:
  username: harry
  password: foobar
  sudo: True
  tty: True
  term_type: xterm
  term_size: (80, 24)
```

2.2 Roster Defaults

If you need to set roster options to be used for all hosts you are targeting you can set *roster_defaults* in the *heist* configuration file. The heist configuration file by default is at /etc/heist/heist.conf.

```
{"roster_defaults":
  {"username": "root"}}
```

2.3 List of Available Rosters

2.3.1 roster modules

heist.roster.clustershell

heist.roster.flat

The default Heist roster if the flat roster.

heist.roster.scan

2.4 Roster Data

Heist allows you to pass in roster data without using a pre-definied roster file. The config *roster-data* allows a user to pass in json data to be used for the roster data.

```
heist salt.minion --roster-data='{"test1":{"host":"192.168.1.2","username":"root",
↪"password":"hostpasswd"}}'
```

The above command will use the host, username and password defined in the json data passed in with *–roster-data*. If you also use *roster-file* alongside *–roster-data*, heist will write the roster data to the specified roster file. For example:

```
heist salt.minion --roster-data='{"test1":{"host":"192.168.1.2","username":"root",
↪"password":"hostpasswd"}}' --roster-file=/tmp/heist-roster
```

This will write the data from *–roster-data* to the file /tmp/heist-roster.

CHAPTER

THREE

HEIST SUBSYSTEMS

Since Heist is built on POP, a Python-based implementation of Plugin Oriented Programming (POP), it is composed of several plugin subsystems, or `subs`. Each `sub` is loaded using POP's dynamic names interface and therefore can be extended using vertical app-merging or adding additional plugins directly to Heist.

3.1 Heist Subsystem

The Heist subsystem is used to create managers for specific daemons. Therefore if there is another agent that someone wanted to add to Heist to make it disolvable and distributable via Heist they would add a plugin to the Heist subsystem.

The required functions to add a new managed agent to Heist are:

3.1.1 run

This is the entry function. The `run` function is used to start the process of creating tunnels and sending daemon code to target systems.

3.1.2 deploy

The `deploy` function is used to deploy the desired code down to the target systems.

3.1.3 update

The `update` function is used to send an updated version of the desired code down to the target system.

3.1.4 clean

The `clean` function is called when Heist gets shut down. This is used to send commands to the remote systems to shut down and clean up the agents.

3.2 Roster Subsystem

The roster subsystem is used to add ways to load up target system data. If it is desired to load roster data from an alternative source a roster can be easily added.

Rosters are very simple. They just need a single async function:

3.2.1 read

The `read` function is called to read in the roster data and returns the roster data structure. The roster data structure is a python dict following this structure:

```
hostname/id:
  logincred: data
  logindata: data
hostname/id:
  logincred: data
  logindata: data
```

3.3 Artifact Subsystem

The artifact system allows for code artifacts that will be deployed to target systems to be downloaded from an artifact source. The artifact source will be specific to the code that is being deployed. It is typical that an artifact plugin will be built in concert with a specific Heist plugin.

3.3.1 get_version

Gather the available version data for the artifacts

3.3.2 get_artifact

Download the actual artifact and store it locally so it can be sent down with the Heist subsystem.

3.4 Tunnel Subsystem

The tunnel subsystem is used to establish communication tunnels with target systems. If you want to use a system for tunneling other than SSH, or you want to use a different SSH backend, just make a new tunnel plugin! The tunnel plugin needs to be able to connect to remote systems, make network tunnels, copy files and execute commands.

3.4.1 create

Used to create the new tunnel instance on the hub. This is where the persistent connection or re-connection (if needed) logic is created.

3.4.2 send

The ability to send files to the target system is implemented here.

3.4.3 get

The ability to retrieve files from the target system is set up here.

3.4.4 cmd

This function runs shell commands on the target system

3.4.5 tunnel

This function creates a network tunnel from ports on the target system back to ports on the source system

3.4.6 destroy

Properly destroy a connection.

CHAPTER

FOUR

CHANGELOG

The Heist project uses the towncrier tool to manage the CHANGELOG.md file. This tool helps manage the changelog and helps prevent merge conflicts when managing one file. This tool adds changelog entries into separate files. Before a release we simply need to run `towncrier --version=<version>` for it to compile the changelog correctly.

4.1 How do I add a changelog entry

To add a changelog entry you will need to add a file in the `changelog` directory. The file name should follow the syntax `<issue #>.<type>`.

The types are in alignment with keepachangelog:

removed:
: any features that have been removed

deprecated:
: any features that will soon be removed

changed:
: any changes in current existing features

fixed:
: any bug fixes

added:
: any new features added

For example, if you are fixing a bug for issue number #1234, your filename would look like this: changelog/1234.fixed. The contents of the file should contain a summary of what you are fixing. If there is a legitimate reason not to include an issue number with a given contribution you can add the MR number as the file name (`<MR #>.<type>`).

If your MR does not align with any of the types, then you do not need to add a changelog entry.

4.2 How to generate the changelog

This step is only used when we need to generate the changelog right before a release. You should NOT run Towncrier on your MR, unless you are preparing the final MR to update the changelog before a release.

You can run the Towncrier tool directly or you can use Nox to run the command and ensure Towncrier is installed in a virtual environment. The instructions below will detail both approaches.

If you want to see what output Towncrier will produce before generating the change log you can run Towncrier in draft mode:

```
towncrier --draft --version=3001
```

```
nox -e 'changelog(draft=True)' -- 3000.1
```

Version will need to be set to whichever version we are about to release. Once you are confident the draft output looks correct you can then generate the changelog by running:

```
towncrier --version=3001
```

```
nox -e 'changelog(draft=False)' -- 3000.1
```

After this is run, Towncrier will automatically remove all the files in the changelog directory.

CHAPTER

FIVE

MANAGING ARTIFACTS WITH HEIST

5.1 Artifact Upgrades

By default, Heist will not upgrade the artifact on a target. If you want to ensure the artifact is always upgraded set *dynamic_upgrade*. If this argument is set, the artifact will be updated in two situations:

- On startup of Heist, if a target already has an artifact deployed it will upgrade if it detects a newer version.
- While the Heist process is running, it will attempt to check if there is a newer version and will upgrade the artifact. This check is done, by default every 60 seconds, but can be adjusted with the conf *checkin_time*.

CHAPTER

SIX

SERVICE PLUGIN

Heist can use a system manager to manage the service of the artifact. By default it will not use a service manager and start the service in the background. If you want to use a service manager you have a couple of options:

1. Set `service_plugin` to the service manager you want to use on the target host in the Heist config.

```
heist:
  service_plugin: systemd
```

2. Set `auto_service` to `True` in Heist's config and Heist will auto detect the system manager.

```
heist:
  auto_service: True
```

3. Set `service_plugin` in your roster file to the service manager you want to use on the target host.

```
system_name:
  host: 10.0.0.28
  username: root
  password: "password"
  service_plugin: systemd
```

Note: The only service managers currently supported are:

- systemd
- win_service

CHAPTER

SEVEN

HEIST SUPPORT ON WINDOWS

Heist uses the SSH protocol to create the communication tunnel to systems it manages. SSH must be installed and listening on systems to be managed with Heist.

Starting with Windows 10 build 1809 and Windows Server 2019, OpenSSH is included as an Optional Feature. Follow the instructions from Microsoft to enable OpenSSH via Windows Settings or Powershell.

On older versions of Windows you'll need to install OpenSSH from a third-party vendor.

If you are provisioning new systems to be managed by Heist, you will need to create a script that will do the following:

- Install or Enable OpenSSH
- Open Firewall Ports
- Start the OpenSSH Service

This script should be set to run when the machine is provisioned. Most cloud and vm providers have methods for mechanisms a script on first boot.

Lucky for you, we have created a powershell script for you! Feel free to modify this script as needed. This script will install OpenSSH using the Optional Features on newer versions of Windows. On older versions without the Optional Features it will install OpenSSH from the Win32-OpenSSH project. The script will also open the firewall and start the SSH service. The script can be found here.

CHAPTER

EIGHT

CONFIGURING HEIST

The configuration file for Heist is located at `/etc/heist/heist.conf` on Linux and `C:\ProgramData\heist\heist.conf` on Windows by default. The `C:\ProgramData` portion of the path might be different in your environment if your `ProgramData` environment variable is set to a different path. If the `ProgramData` environment variable is not set on Windows, then Heist will use `C:\ProgramData` by default.

The heist.conf file contents need to be under a `heist` yaml dictionary. For example:

```
heist:
  dynamic_upgrade:True
  checkin_time: 5
```

You can change the location of the Heist configuration file two ways:

8.1 cli opts

8.1.1 -c

You can pass `-c /opt/heist.conf` on the cli when running Heist. This example would use the file `/opt/heist.conf` for the Heist configuration file.

8.1.2 –manage-service

Heist can manage the service of an artifact that was previously deployed. The allowed options are start, stop, restart, status, enable and disable. This argumment will manage the service and then close the Heist connection.

```
heist <manager> -R /etc/heist/roster --manage-service=start
```

8.1.3 –clean

If there is a previously deployed artifact on the target, Heist will clean the artifact before re-deploying again. If there was not a previous artifact deployed it will log an error but continue deploying a new artifact.

```
heist <manager> -R /etc/heist/roster --clean
```

8.1.4 environment variable

You can set the environment variable `HEIST_CONFIG` to the path of the configuration you want to use for Heist.

8.2 Primary Heist Configuration

8.2.1 acct_profile

Default: `default`

The specified named profile to read from encrypted acct files

```
heist:
  acct_profile: testprofile
```

8.2.2 artifacts_dir

Linux Default: `/var/tmp/heist/artifacts` Windows Default: `C:\\ProgramData\\heist\\artifacts`

The location to look for artifacts that will be sent to target systems

```
heist:
  artifacts_dir: /etc/artifacts/
```

8.2.3 roster

Default: `None`

The type of roster to use to load up the remote system to tunnel into. If the file extension of the roster file is `.fernet` the default roster will be the `fernet` roster. Otherwise, the default is the `flat` roster.

```
heist:
  roster: scan
```

8.2.4 roster_dir

Linux Default: `/etc/heist/rosters` Windows Default: `C:\\ProgramData\\heist\\rosters`

The directory to look for roster files when using the `flat` roster.

```
heist:
  roster_dir: /var/rosters
```

8.2.5 roster_file

Linux Default: `/etc/heist/roster` Windows Default: `C:\\ProgramData\\heist\\roster`

Use a specific roster file. When using the `flat` roster if this option is not used, then the `roster_dir` will be used to find roster files.

```
heist:
  roster_file: /var/heist/roster
```

8.2.6 checkin_time

Default: 60

The number of seconds between checking to see if the managed systems need to get an updated binary or agent restart.

```
heist:
  checkin_time: 100
```

8.2.7 dynamic_upgrade

Default: `False`

Heist will detect when new binaries are available and dynamically upgrade the target systems.

```
heist:
  dynamic_upgrade: True
```

8.2.8 renderer

Default: `yaml`

Specify the renderer to use to render heist roster files.

```
heist:
  renderer: toml
```

8.2.9 target

Default: `None`

Target used for multiple rosters. This argument is required for some rosters such as scan and clustershell.

```
heist:
  target: 10.0.0.2
```

8.2.10 artifact_version

Default: `None`

Version of the artifact to use for heist

```
heist:
  artifact_version: 3005
```

8.2.11 roster_defaults

Default: `{}`

Default options to use for all rosters. CLI options will override these defaults.

```
heist:
  roster_defaults:
      username: testuser
```

8.2.12 service_plugin

Default: `raw`

The type of service to use when managing the artifacts service status.

```
heist:
  service_plugin: systemd
```

8.2.13 auto_service

Default: `False`

Attempt to auto detect the service manager to use on start up of service.

```
heist:
  auto_service: True
```

8.2.14 noclean

Default: `False`

If set to `True` do not clean the artifact and configs on the target. If `False`, the artifact and configs will be removed from the target.

```
heist:
  noclean: True
```

8.2.15 run_dir_root

Default: `False`

Directory location on remote system for root deployment.

```
heist:
  run_dir_root: /opt/run/
```

CHAPTER

NINE

HEIST CONFIGURATION

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search