cowrie Documentation

Release 3.0.4.dev42+gd610dc061

Michel Oosterhof

Jun 21, 2026

CONTENTS:

CHAPTER

ONE

COWRIE

1.1 What is Cowrie

Cowrie is a medium to high interaction SSH and Telnet honeypot designed to log brute force attacks and the shell interaction performed by the attacker. In medium interaction mode (shell) it emulates a UNIX system in Python, in high interaction mode (proxy) it functions as an SSH and telnet proxy to observe attacker behavior to another system. In LLM mode, it uses large language models to generate dynamic responses to attacker commands.

Cowrie is maintained by Michel Oosterhof.

1.2 Documentation

The Documentation can be found here.

1.3 Slack

You can join the Cowrie community at the following Slack workspace.

1.4 Features

- **Choose to run as an emulated shell (default):**
  - Fake filesystem with the ability to add/remove files. A full fake filesystem resembling a Debian 5.0 installation is included
  - Possibility of adding fake file contents so the attacker can *cat* files such as */etc/passwd*. Only minimal file contents are included
  - Cowrie saves files downloaded with wget/curl or uploaded with SFTP and scp for later inspection
- **Or proxy SSH and telnet to another system**
  - Run as a pure telnet and ssh proxy with monitoring
  - Or let Cowrie manage a pool of QEMU emulated servers to provide the systems to login to
- **Or use an LLM backend (experimental):**
  - Use large language models (e.g., OpenAI GPT) to dynamically generate realistic shell responses
  - Handles any command without predefined responses
  - Maintains conversation context for consistent sessions

For both settings:

- Session logs are stored in a UML Compatible format for easy replay with the *playlog* utility.
- SFTP and SCP support for file upload
- Support for SSH exec commands
- Logging of direct-tcp connection attempts (ssh proxying)
- Forward SMTP connections to SMTP Honeypot (e.g. mailoney)
- JSON logging for easy processing in log management solutions

1.5 Installation

There are currently three ways to install Cowrie: *git clone*, *Docker* and *pip*. *Docker* is the easiest to try and run, but to configure and modify you'll need a good understanding of containers and volumes. *git clone* is recommended if you want to change the configuration of the honeypot. *pip* mode is still under development.

1.6 Docker

Docker images are available on Docker Hub.

- To get started quickly and give Cowrie a try, run:

```
$ docker run -p 2222:2222 cowrie/cowrie:latest
$ ssh -p 2222 root@localhost
```

- To just make it locally, run:

```
$ make docker-build
```

1.7 PyPI

Cowrie is available on PyPI, to install run:

```
$ pip install cowrie
$ twistd cowrie
```

When installed this way, it will behave differently from having a full directory download.

This is still in beta and may not work as expected, *git clone* or *docker* methods are preferred.

1.8 Requirements

Software required to run locally:

- Python 3.10+
- python-virtualenv

1.9 Files of interest:

- *etc/cowrie.cfg* - Cowrie's configuration file (operator-owned). Created by `cowrie init`.
- src/cowrie/data/etc/cowrie.cfg.dist - bundled defaults, edit your `etc/cowrie.cfg` instead

- *etc/userdb.txt* - credentials to access the honeypot
- *src/cowrie/data/fs.pickle* - fake filesystem; carries both metadata (path, uid, gid, size, mode) and the embedded contents (`A_CONTENTS` bytes) for the small files attackers commonly cat. Edit via `fsctl`; rebuild via `make build-fs-pickle`.
- *src/cowrie/data/txtcmds/* - output for simple fake commands
- *var/log/cowrie/cowrie.json* - audit output in JSON format
- *var/log/cowrie/cowrie.log* - log/debug output
- *var/lib/cowrie/tty/* - session logs, replayable with the *playlog* utility.
- *var/lib/cowrie/downloads/* - files transferred from the attacker to the honeypot are stored here

1.10 Commands

- *cowrie* - start, stop and restart Cowrie
- *fsctl* - modify the fake filesystem
- *createfs* - create your own fake filesystem
- *playlog* - utility to replay session logs
- *asciinema* - turn Cowrie logs into asciinema files

1.11 Contributors

Many people have contributed to Cowrie over the years. Special thanks to:

- Upi Tamminen (desaster) for all his work developing Kippo on which Cowrie was based
- Dave Germiquet (davegermiquet) for TFTP support, unit tests, new process handling
- Olivier Bilodeau (obilodeau) for Telnet support
- Ivan Korolev (fe7ch) for many improvements over the years.
- Florian Pelgrim (craneworks) for his work on code cleanup and Docker.
- Guilherme Borges (sgtpepperpt) for SSH and telnet proxy (GSoC 2019)
- And many many others.

CHAPTER

TWO

CONTRIBUTING GUIDELINES

Thank you for your interest in contributing to our project. Whether it's a bug report, new feature, correction, or additional documentation, we greatly value feedback and contributions from our community.

Please read through this document before submitting any issues or pull requests to ensure we have all the necessary information to effectively respond to your bug report or contribution.

2.1 Reporting Bugs/Feature Requests

We welcome you to use the GitHub issue tracker to report bugs or suggest features.

When filing an issue, please check existing open, or recently closed, issues to make sure somebody else hasn't already reported the issue. Please try to include as much information as you can. Details like these are incredibly useful:

- A reproducible test case or series of steps
- The version of our code being used
- Any modifications you've made relevant to the bug
- Anything unusual about your environment or deployment

2.2 Contributing via Pull Requests

Contributions via pull requests are much appreciated. Before sending us a pull request, please ensure that:

1. You are working against the latest source on the *main* branch.
2. You check existing open, and recently merged, pull requests to make sure someone else hasn't addressed the problem already.
3. You open an issue to discuss any significant work - we would hate for your time to be wasted.

To send us a pull request, please:

1. Fork the repository.
2. Modify the source; please focus on the specific change you are contributing. If you also reformat all the code, it will be hard for us to focus on your change.
3. Ensure local tests pass.
4. Commit to your fork using clear commit messages.
5. Send us a pull request, answering any default questions in the pull request interface.
6. Pay attention to any automated CI failures reported in the pull request, and stay involved in the conversation.

GitHub provides additional document on forking a repository and creating a pull request.

2.3 Finding contributions to work on

Looking at the existing issues is a great way to find something to contribute on. As our projects, by default, use the default GitHub issue labels ((enhancement/bug/duplicate/help wanted/invalid/question/wontfix), looking at any help wanted issues is a great place to start.

2.4 Licensing

See the LICENSE file for our project's licensing. We will ask you confirm the licensing of your contribution.

CHAPTER

THREE

LICENSE

CHAPTER

FOUR

FREQUENTLY ASKED QUESTIONS

4.1 Do I need to copy all the content of cowrie.cfg.dist to cowrie.cfg?

No, Cowrie merges your local settings in `cowrie.cfg` and the default settings will automatically be read from `cowrie.cfg.dist`

4.2 Why certain commands aren't implemented?

Implementing all possible UNIX commands in Python is not worth the time and effort. Cowrie tries to provide most common commands used by attackers of the honeypot. If you see attackers use a command that you'd like to see implemented, please let us know, or send a pull request.

4.3 How do I add or modify the default user?

The default Cowrie user is called *phil* these days. Having the same user always available is an easy way to identify Cowrie, so it's recommended to change this. The bundled defaults are baked into `src/cowrie/data/fs.pickle` as `A_CONTENTS` bytes — either edit a copy of the pickle directly, or set `[honeypot] contents_path` in `etc/cowrie.cfg` and drop an override file at `<contents_path>/etc/passwd`.

For the per-file-override path:

```
[honeypot]
contents_path = /opt/cowrie/honeyfs
```

And then:

```
$ mkdir -p /opt/cowrie/honeyfs/etc
$ cp /path/to/your/passwd /opt/cowrie/honeyfs/etc/passwd
```

Rename the user in the filesystem tree too:

```
$ fsctl src/cowrie/data/fs.pickle
fs.pickle:/$ mv /home/phil /home/joe
```

(For a custom copy of the pickle, copy it out first, set `[shell] filesystem` to point at it, and edit there — see INSTALL.rst's "Customising the honeypot" section.)

And then restart Cowrie:

```
(cowrie-env) $ cowrie restart
```

4.4 How do I add files to the file system?

The filesystem metadata and embedded contents both live in `src/cowrie/data/fs.pickle`. Adding a new path makes it show up in `ls` and other commands; you can populate its contents the same way.

Use `fsctl` to add the filesystem entry and load its contents:

```
(cowrie-env) $ fsctl src/cowrie/data/fs.pickle
fs.pickle:/$ touch /home/phil/myfile 1024
fs.pickle:/$ chown 1000:1000 /home/phil/myfile
fs.pickle:/$ load /home/phil/myfile /local/path/to/myfile
fs.pickle:/$ exit
```

For bulk content updates (e.g. loading every file under a local directory tree), use `fsctl <pickle> "embed <local-dir>"`.

CHAPTER

FIVE

INSTALLING COWRIE

Cowrie supports two install paths:

- **Pip install** (recommended for operators). Install the published package, `cowrie init` a state directory, edit `etc/cowrie.cfg`, start. No source checkout required.
- **Source checkout** (developers and contributors). Clone the repo and install in editable mode.

For `proxy` mode, see `PROXY.rst`.

5.1 Contents

5.2 Quick start: pip install

For most operators, this is the shortest path:

```
$ mkdir ~/my-honeypot && cd ~/my-honeypot
$ python3 -m venv cowrie-env
$ source cowrie-env/bin/activate
(cowrie-env) $ pip install cowrie
```

(continues on next page)

(continued from previous page)

```
(cowrie-env) $ cowrie init
(cowrie-env) $ $EDITOR etc/cowrie.cfg     # optional
(cowrie-env) $ cowrie start
```

The venv lives inside the honeypot directory alongside `etc/` and `var/`, keeping each honeypot self-contained.

The pip-install workflow described here requires Cowrie 3.0.0 or later (or the current `main` branch). Earlier releases need the source checkout path below.

`cowrie init` writes `./etc/cowrie.cfg` from the bundled template. On first `cowrie start` cowrie creates the rest of the state layout (`var/log/cowrie/`, `var/lib/cowrie/`, `var/run/`) under the same directory and generates SSH host keys. Pick the directory you want state to live in before running `init`.

Read on for system-dependency setup, port-22 listening, and other optional pieces.

5.2.1 Step 1: System dependencies

Cowrie itself is pure Python, but several of its dependencies have native components (`cryptography`, `cffi`, `bcrypt`, optional `mysqlclient` / `libvirt-python`). On most distros you need build-essential plus the OpenSSL and libffi headers for these to compile during `pip install`.

On Debian-based systems (last verified on Debian Bookworm):

```
$ sudo apt-get install python3-pip python3-venv libssl-dev libffi-dev build-essential␣
↪libpython3-dev python3-minimal authbind
```

For a source checkout, additionally install:

```
$ sudo apt-get install git docker.io
```

(`docker.io` is only needed if you want to rebuild `fs.pickle` from a Debian container via `make build-fs-pickle`.)

5.2.2 Step 2: Create a user account

It is strongly recommended to run Cowrie as a dedicated non-root user:

```
$ sudo adduser --disabled-password cowrie
$ sudo su - cowrie
```

Cowrie refuses to start as root.

5.2.3 Step 3: Install Cowrie

5.3 Pip install (operators)

Pick the directory you want cowrie state (logs, downloads, host keys, ttylogs) to live in, create the venv inside it, and install:

```
$ mkdir ~/my-honeypot && cd ~/my-honeypot
$ python3 -m venv cowrie-env
$ source cowrie-env/bin/activate
(cowrie-env) $ python -m pip install --upgrade pip
(cowrie-env) $ python -m pip install cowrie
```

The venv is kept alongside `etc/` and `var/` so the honeypot directory is self-contained.

5.4 Source checkout (developers)

If you plan to modify Cowrie or run against unreleased code, you also need the development extras (mypy, ruff, pre-commit, tox, sphinx, etc.) and a few extra system packages for native builds:

```
$ sudo apt-get install git docker.io
$ git clone https://github.com/cowrie/cowrie
$ cd cowrie
$ python3 -m venv cowrie-env
$ source cowrie-env/bin/activate
(cowrie-env) $ python -m pip install --upgrade pip
(cowrie-env) $ python -m pip install -e '.[dev]'
```

`docker.io` is only required if you want to use `make build-fs-pickle` to regenerate the bundled filesystem from a Debian container. The `[dev]` extra brings in the typecheckers, linters, test runner, and docs toolchain that match what CI uses.

In source-checkout mode, the repo root *is* the state directory. `cowrie start` detects this and skips the `cowrie init` step.

5.4.1 Step 4: Initialise the state directory

(Skip this step if you are using a source checkout.)

From inside the honeypot directory you created in Step 3, run:

```
(cowrie-env) $ cowrie init
Wrote etc/cowrie.cfg
Created var/log/cowrie, var/lib/cowrie, var/lib/cowrie/downloads, var/lib/cowrie/tty,␣
↪var/run
Edit etc/cowrie.cfg to customise hostname, ports, etc., then run `cowrie start`.
```

`cowrie init` writes `./etc/cowrie.cfg` from the bundled template and creates the `var/` skeleton so the first `cowrie start` has somewhere to write logs and a PID file. SSH host keys are generated on first start.

If the config already exists, `cowrie init` refuses with a non-zero exit code rather than overwriting your edits — re-running `cowrie init` is *not* idempotent.

5.4.2 Step 5: Configure

Configuration lives in `./etc/cowrie.cfg` relative to the directory you run Cowrie from. The full set of available settings and their defaults are documented in the bundled `cowrie.cfg.dist` (also materialised by `cowrie init` for browsing).

Cowrie loads configuration in layers:

1. Bundled defaults (the `cowrie.cfg.dist` shipped inside the package).
2. `/etc/cowrie/cowrie.cfg` (system-wide install, if present).
3. `./etc/cowrie.cfg` (per-state-directory).
4. `./cowrie.cfg` (alternate flat layout).

Later layers override earlier ones for any keys they set. Your `etc/cowrie.cfg` only needs to contain the keys you want to change.

To enable Telnet, for example, the entire `cowrie.cfg` could be:

```
[telnet]
enabled = true
```

5.4.3 Step 6: Start Cowrie

```
(cowrie-env) $ cowrie start
```

Cowrie runs in the current working directory. Logs land in `./var/log/cowrie/` and the PID file is `./var/run/cowrie.pid`.

`cowrie start` refuses to run from a directory that has not been initialised. If you see `ERROR: cowrie is not initialised` you are probably in the wrong directory — `cd` into your state directory first, or run `cowrie init`.

5.4.4 Step 7: Listening on port 22 (OPTIONAL)

The SSH daemon runs on port 22 by default. Cowrie defaults to port 2222. To collect most traffic, you need Cowrie listening on 22. This requires two changes: relocate any existing SSH server, then expose Cowrie on the lower port.

There are three approaches: `iptables` redirection, `authbind`, or `setcap`.

5.5 Iptables

Port redirection is system-wide and runs as root. A firewall redirect can make your existing SSH server unreachable — move it to a different port first.

Linux:

```
$ sudo iptables -t nat -A PREROUTING -p tcp --dport 22 -j REDIRECT --to-port 2222
```

With nft:

```
$ sudo nft add rule ip nat prerouting tcp dport 22 redirect to 2222
```

Telnet equivalents:

```
$ sudo iptables -t nat -A PREROUTING -p tcp --dport 23 -j REDIRECT --to-port 2223
$ sudo nft add rule ip nat prerouting tcp dport 23 redirect to 2223
```

Note: test from another host — these rules do not apply to loopback.

macOS:

```
$ echo "rdr pass inet proto tcp from any to any port 22 -> 127.0.0.1 port 2222" | sudo␣
↪pfctl -ef -
```

5.6 Authbind

Run Cowrie as a non-root user but bind directly to port 22:

```
$ sudo apt-get install authbind
$ sudo touch /etc/authbind/byport/22
$ sudo chown cowrie:cowrie /etc/authbind/byport/22
$ sudo chmod 770 /etc/authbind/byport/22
```

Set the listening port in `etc/cowrie.cfg`:

```
[ssh]
listen_endpoints = tcp:22:interface=0.0.0.0
```

Or for Telnet:

```
$ sudo touch /etc/authbind/byport/23
$ sudo chown cowrie:cowrie /etc/authbind/byport/23
$ sudo chmod 770 /etc/authbind/byport/23
```

And in `etc/cowrie.cfg`:

```
[telnet]
listen_endpoints = tcp:23:interface=0.0.0.0
```

Start with authbind enabled:

```
$ AUTHBIND_ENABLED=yes cowrie start
```

5.7 Setcap

Grant the Python binary the bind-low-port capability:

```
$ setcap cap_net_bind_service=+ep /usr/bin/python3
```

Then change the listen ports in `etc/cowrie.cfg` as above.

5.7.1 Installing Backend Pool dependencies (OPTIONAL)

If you want proxy mode with the automatic backend pool, install QEMU and libvirt:

```
$ sudo apt-get install qemu-system-arm qemu-system-x86 libvirt-dev libvirt-daemon␣
↪libvirt-daemon-system libvirt-clients nmap
```

Then install the Python extra:

```
(cowrie-env) $ python -m pip install 'cowrie[pool]'
```

(In a source checkout: `python -m pip install -e '.[pool]'`.)

To let QEMU use disk images and snapshots, edit `/etc/libvirt/qemu.conf` and set both `user` and `group` to the user running the pool (typically `cowrie`).

5.7.2 Running using supervisord (OPTIONAL)

In `/etc/supervisor/conf.d/cowrie.conf`:

```
[program:cowrie]
command=/home/cowrie/cowrie-env/bin/cowrie start -n
directory=/home/cowrie/my-honeypot
user=cowrie
autorestart=true
redirect_stderr=true
```

The `directory=` must point at the state directory you initialised in Step 4.

5.7.3 Configure Additional Output Plugins (OPTIONAL)

Cowrie automatically outputs event data to text and JSON log files in `./var/log/cowrie`. Additional output plugins can record the data elsewhere. Supported plugins include:

- Cuckoo
- ELK (Elastic) Stack
- Graylog
- Splunk
- SQL (MySQL, SQLite3, RethinkDB)

See `docs/[Output Plugin]/README.rst` for details.

5.7.4 Troubleshooting

5.8 cowrie is not initialised in this directory

You ran `cowrie start` from a directory that does not look like a cowrie state directory. Either `cd` to your state directory first, or run `cowrie init` to set up the current directory.

5.9 CryptographyDeprecationWarning: Blowfish has been deprecated

Safe to ignore:

```
CryptographyDeprecationWarning: TripleDES has been moved to ...
```

5.10 twistd: unknown command: cowrie

Two possibilities. If there is a Python stack trace, a dependency is missing or broken. Without a stack trace, double-check that you activated the right virtualenv.

5.11 General approach

Check the log file `./var/log/cowrie/cowrie.log` (relative to wherever you started Cowrie).

5.11.1 Updating Cowrie

Cowrie commands operate on the current working directory — the PID file, log paths, and state files are all relative to wherever you ran `cowrie start` from. Stop and start commands must be issued from the same directory.

Stop your honeypot first:

```
(cowrie-env) $ cd ~/my-honeypot
(cowrie-env) $ cowrie stop
```

Pip install:

```
(cowrie-env) $ python -m pip install --upgrade cowrie
```

Source checkout:

```
(cowrie-env) $ cd ~/cowrie
(cowrie-env) $ git pull
(cowrie-env) $ python -m pip install --upgrade -e .
```

If you use the SQL/Splunk/ELK output plugins, also upgrade their optional dependencies:

```
(cowrie-env) $ python -m pip install --upgrade -r requirements-output.txt
```

Restart:

```
(cowrie-env) $ cd ~/my-honeypot
(cowrie-env) $ cowrie start
```

5.11.2 Customising the honeypot

The simulated filesystem and the default file contents that attackers see (`/etc/passwd`, `/etc/hostname`, `/proc/cpuinfo`, etc.) ship inside the bundled `fs.pickle`. Three customisation paths:

5.12 Per-file operator override

Set `contents_path` in `etc/cowrie.cfg` to a directory of your choice, and drop a file at the matching path inside it:

```
[honeypot]
contents_path = /opt/cowrie/honeyfs
```

Then:

```
$ mkdir -p /opt/cowrie/honeyfs/etc
$ vi /opt/cowrie/honeyfs/etc/issue.net
```

Files present in this directory override the bundled defaults at matching paths. Anything not overridden continues to come from the pickle. Only files that already exist in the pickle are visible to attackers via `cat`; adding a brand-new path requires editing the pickle itself.

5.13 Editing the pickle

The bundled `fs.pickle` lives inside the installed Cowrie package and is read-only from an operator's perspective. To edit it, copy it out first and point `[shell] filesystem` at your local copy:

```
(cowrie-env) $ python -c "from cowrie.core.resources import read_data_bytes; \
    open('var/lib/cowrie/fs.pickle', 'wb').write(read_data_bytes('fs.pickle'))"
```

Add to `etc/cowrie.cfg`:

```
[shell]
filesystem = var/lib/cowrie/fs.pickle
```

Then edit with `fsctl`. For a one-off file change:

```
$ fsctl var/lib/cowrie/fs.pickle "load /etc/passwd /path/to/your/passwd"
```

For a bulk update of many files from a local directory:

```
$ fsctl var/lib/cowrie/fs.pickle "embed /path/to/your/honeyfs"
```

Use `fsctl <pickle>` with no command to enter the interactive shell.

5.14 Rebuilding the pickle from scratch

To regenerate the bundled filesystem from a fresh Debian container with custom packages, see `bin/build-fs-pickle.sh` and the `make build-fs-pickle` target. Paths listed in `createfs.py`'s `EMBED_PATHS` have their bytes baked into the pickle during the build.

5.14.1 Local testing with SSH

After starting Cowrie, you can test by connecting via SSH:

1. Start Cowrie:

   ```
   (cowrie-env) $ cd ~/my-honeypot
   (cowrie-env) $ cowrie start
   ```

2. Connect (Cowrie listens on 2222 by default):

   ```
   $ ssh -p 2222 root@localhost
   ```

3. If authentication fails, check `etc/userdb.txt` and confirm an allow rule exists. Rules are processed top-to-bottom and stop at the first match.

4. Logs land in `./var/log/cowrie/cowrie.log` — all recorded activity, login attempts, and shell commands.

CHAPTER

SIX

RELEASE NOTES

6.1 Release 3.0.0

BREAKING CHANGES - ACTION REQUIRED:

- **State directory layout is now cwd-driven.** `cowrie start` no longer `chdir`s to a script-derived "root" path. The current working directory is the cowrie state directory: `./etc/cowrie.cfg` is the config, `./var/log/cowrie/` holds logs, `./var/run/` holds the PID file. Run all cowrie commands (`start`, `stop`, `restart`, `status`) from the same directory.
- **New ``cowrie init`` command.** Pip-install operators run `cowrie init` once in their chosen state directory to materialise `./etc/cowrie.cfg` from the bundled template and create the `var/{log/cowrie,lib/cowrie,run}` skeleton. Source-checkout users unaffected — the repo root counts as initialised via `src/cowrie/data/etc/cowrie.cfg.dist`.
- **``cowrie start`` refuses to run without an init marker.** Looks for one of `./etc/cowrie.cfg`, `./etc/cowrie.cfg.dist`, or `./src/cowrie/data/etc/cowrie.cfg.dist`. Prevents accidentally polluting the wrong directory with state files.
- **Top-level ``honeyfs/`` directory removed.** Its contents are now embedded in `src/cowrie/data/fs.pickle` as `A_CONTENTS` bytes. Operators with local customisations: keep your honeyfs files in a directory of your choice and set `[honeypot] contents_path` to it in `cowrie.cfg`; per-file overrides still cascade on top of the bundled defaults.
- **Bundled ``cowrie.cfg.dist`` moved into the package.** The template is now at `src/cowrie/data/etc/cowrie.cfg.dist` and is loaded as the defaults layer automatically. Your operator config (`etc/cowrie.cfg`) only needs to contain the keys you want to override.
- **``[honeypot] data_path`` config key removed.** Settings that used `${honeypot:data_path}/...` (`filesystem`, `processes`, `config_files_path`) now read their bundled defaults from the package directly when left unset. Operators who set `data_path` in their cowrie.cfg should set each derived path individually instead.
- **``[honeypot] contents_path`` is now unset by default.** Previously defaulted to `honeyfs` (the removed top-level directory). Leaving it unset serves all file contents from the pickle. Set it only when you want per-file overrides.

CONFIGURATION CHANGES:

- Bundled `cowrie.cfg.dist` has `contents_path`, `filesystem`, and `processes` commented out (each documents how to override).
- `data_path` removed from cfg.dist.

DEPENDENCIES:

- `tftpy` removed from runtime dependencies. The TFTP client (`cowrie/commands/tftp.py`) uses a Twisted `DatagramProtocol` implementation; tftpy was never imported.

INTERNAL:

- New `cowrie/shell/honeyfs.py` module owns the per-process pickle cache. `HoneyPotFilesystem` instances now share a single `pickle.load` via `honeyfs.get_tree()` (deepcopied per session) instead of re-reading from disk on every connection.
- New `cowrie/core/resources.py` provides `read_data_bytes` and `open_data_binary` for accessing bundled `cowrie.data` resources via `importlib.resources`.
- `createfs.py` grew an `EMBED_PATHS` constant — paths whose bytes are baked into `A_CONTENTS` during the recursive walk.
- `fsctl` gained an `embed <local-dir>` command for bulk-loading file contents into an already-built pickle.
- `Passwd` and `Group` classes moved from class-body load to instance `__init__`.
- `backend_pool` XML config templates now use a bundled-data cascade matching the honeyfs pattern. `[backend_pool] config_files_path` is optional; unset falls through to bundled defaults via `importlib.resources`.
- CI smoke tests for PyPI packages replaced the no-op `twistd cowrie` invocation with checks that verify the entry point, bundled resources, `cowrie init` output, and the init-marker guard on `cowrie start`.

6.2 Releases 2.9.1 – 2.9.20

NEW FEATURES:

- **New shell commands**: `cut`.
- **IPv6 support**: `ifconfig` and `netstat` now show Global Unicast Addresses; `get_endpoints_from_section` detects IPv6 listen addresses.
- **LLM proxy support**: the LLM backend reads `HTTP_PROXY` / `HTTPS_PROXY` environment variables for outbound requests.
- **LLM Anthropic provider**: Claude can now be used as an LLM backend alongside OpenAI. The context prompt sent to the model is configurable.
- **Shell script execution**: scripts created via output redirection (e.g. `cat > script.sh`) can now be executed.
- **CVE-2026-24061 detection**: telnet NEW-ENVIRON exploit attempts are detected and logged; the honeypot emulates the vulnerable response.
- **OS fingerprint update**: default emulated OS bumped to Debian 12 / kernel 6.1.
- **SPDX / REUSE compliance**: all source files carry SPDX license headers; `reuse lint` passes in CI.

BUG FIXES:

- **SSRF protection bypass** in `ftpget`, `tftp`, and `nc` commands (reserved IP range checks were bypassable).
- `chmod --help` and `--version` were broken.
- `chattr` command was shadowed by a no-op stub.
- `wget` saved downloaded files to `/` instead of the session's cwd.
- `playlog` now shows output for exec sessions.
- File-descriptor redirection parsing handles more edge cases.
- Tab completion no longer errors on non-existing directories.
- `backend_pool` NAT errors in remote mode when the client protocol was not yet initialised or already closed.
- Telnet infinite recursion when `onResult` callback is `None`.

- `nc` command: more realistic output and abuse-protection limits.

INFRASTRUCTURE:

- Docker base image upgraded to Debian 13 (trixie) / Python 3.13.
- Twisted bumped to 26.4.0 with stricter typing adopted.
- Malshare and Cuckoo output plugins ported from `requests` to `treq`.
- Weekly automated release workflow added.
- Bundled `fs.pickle` now embeds file contents (`A_CONTENTS` bytes).

6.3 Release 2.9.0

NEW FEATURES:

- **LLM Backend**: New experimental backend that uses Large Language Models (such as OpenAI's GPT) to generate realistic shell responses. Instead of static command emulation, the LLM dynamically generates output for any command, making the honeypot more convincing and capable of handling unexpected inputs. See the LLM documentation for setup instructions.

CONFIGURATION CHANGES:

- New `[llm]` configuration section for LLM backend settings including API key, model selection, and response parameters.

6.4 Release 2.7.0

BREAKING CHANGES - ACTION REQUIRED:

- Install Cowrie into your virtual environment with *pip install -e .*
- **bin/ directory removed**: Scripts *asciinema*, *createfs*, *fsctl*, and *playlog* are no longer called from the *bin/* directory.
- **Python 3.9 no longer supported**: Minimum Python version is now 3.10.
- **SQL schema update required**: If using MySQL/SQLite databases, run the migration script *docs/sql/update16.sql* to extend IP address fields for IPv6 support (VARCHAR length increased to 61 characters).
- **SSH-DSS key support removed**: The deprecated ssh-dss algorithm is no longer supported for improved security. Remove ssh-dss configuration if you use it.

NEW FEATURES:

- **New Output Plugins**: * PostgreSQL output plugin with automatic reconnection support * Prometheus metrics output plugin for monitoring and alerting
- **New Shell Commands**: * *find* command with basic options for file searching * *dig* command for DNS lookups * *git* command for version control simulation * *curl* command now supports HEAD requests with *-I* option
- **Enhanced Security**: * Network blocking for outbound connections from wget/curl/nc to reserved IP ranges * Null byte password protection to prevent authentication bypasses * Updated SSH algorithms and key management for better security posture
- **Proxy Mode Improvements**: * SFTP file transfers now logged and captured in proxy mode * Better SSH factory handling for improved stability

CONFIGURATION CHANGES:

- New configuration options available for: * PostgreSQL output plugin settings * Prometheus metrics endpoint configuration * Network blocking controls for command simulation

INFRASTRUCTURE UPDATES:

- **Docker**: * Improved local build support * Container signing with Cosign for supply chain security * Updated base images and metadata
- **Build System**: * Migrated to setuptools-scm for automatic version management * PyPI package publishing now automated as trial for future development
- **Dependencies**: * Twisted updated to 25.5.0 * Elasticsearch client updated to 9.x * Various security updates across all dependencies

IMPROVED FEATURES:

- Enhanced MISP output plugin with overcuriosity protection and better threat intelligence integration
- Simplified Slack output formatting for better readability
- Better shell command substitution and subshell execution
- Improved error handling in wget with explicit timeouts
- ECS-compliant Logstash configuration template
- Enhanced history handling in shell sessions

DEVELOPMENT:

- Added Python 3.14 development version support
- Added PyPy 3.11 support
- Improved test coverage and CI/CD pipelines

6.5 Release 2.6.0

- Breaking change: default location of static files has moved from share/cowrie to src/cowrie/data
- In the configuration file the *share_path* is now *data_path*
- Python 3.12 support
- Python 3.13 support
- Pypy 3.10 support
- Python 3.8 no longer supported
- Twisted 24.10 support
- Docker builds now use Debian 12 Bookworm
- New output plugins: Oracle, Remote Syslog, Axiom
- New commands: finger, groups, locate, lspci
- Cowrie can now be installed with *pip install -e*

6.6 Release 2.5.0

- Datadog output module (Fred Baguelin <frederic.baguelin@datadoghq.com>)
- General improvements to shell expansion handling

- New version of Twisted supported
- Python 3.11 support
- Pypy 3.9 support
- Add session type to Telegram output

6.7 Release 2.4.0

- Deprecate Python 3.7
- Early support for Python 3.11
- ThreatJammer output plugin (@diegoparrilla)
- Telegram output plugin (@Louren)
- Discord output plugin (@CyberSparkNL)
- Updated mongodb output plugin
- Dependency upgrades
- Docker repo merged with this one
- *wget* and *curl* rewritten using *treq*.
- Migrate test framework from trial to unittest (@lazycrazyowl)

6.8 Release 2.3.0

- Deprecate Python 3.6
- Support Python 3.10
- Dependency updates
- MISP Output plugin extension
- add new public keys ECDSAKeys and ed25519 (#1627)
- fix userdb.example (#1619)
- cache url submission to virustotal
- MySQL connector (#1575) - needs new external dependency mysql-connector-python
- Fix mysql string expansion (#1565)
- Rewrite CSIRTG output plugin to use new library version
- Fixed the Slack output to work with the versions 2.x of slackclient
- fix MySQL error handling
- fix tar command
- limit connections to private address ranges
- Update GreyNoise Output Script to Use Community API (#1524)
- Implement getopt-style parsing for uname (#1516)
- Allow SSLv3 connections for wget and curl
- Support for 301 redirects in wget

- Malshare update API (#1472)
- Remove hpfeeds.py infavour of hpfeeds3.py

6.9 Release 2.2.0

- Deprecate Python 2.7 and 3.5
- Command substitution with backticks
- Better `chmod` command line parsing
- Add `uniq` command.
- Enhanced command substitution functionality.
- Fix nc hang
- Rename built-in user `richard` to `phil`, it's used as detection mechanism.
- Binary suppport for `cat`, `grep` and other commands
- Azure Sentinel output plugin

6.10 Release 2.1.0

- Deprecate Python 2.7. Still works but removed from testing suite and fixing 2.7 problems will no longer have priority.
- Disable crashreporter
- Updated ELK documentation and output plugin
- `tee` command added. Updates to `cat`, `dd` and `wc`.
- Fixed SSH compression issue with AsyncSSH client
- AbuseIP output plugin.

6.11 Release 2.0.1

- 2019-10-31 Fix for exec commands when tty logging is disabled
- 2019-10-31 Fix for print output to stdout for curl/wget
- 2019-10-31 Fix for SQL to store full hostname (don't forget to update the database schema)
- 2019-10-15 Slack link now at https://cowrie.org/slack
- 2019-10-04 Subshell ((echo test)) evaluation now working

6.12 Release 2.0.0

- 2019-09-06 Crash reporter is enabled by default and will upload data on crashes to api.cowrie.org. This can be disabled in by setting `enabled=false` in `[output_crashreporter]`
- 2019-09-05 Proxy functionality now active by @sgtpepperpt and GSoC2019
- 2019-06-20 Move *auth_none* and *auth_keyboard_interactive_enabled* to [ssh] config section

6.13 Release 1.6.0

- 2019-03-31 New documentation theme
- 2019-03-23 Greynoise output plugin (@mzfr)
- 2019-03-19 direct-tcp forwarding now written to databases (@gborges)
- 2019-03-19 Reverse DNS output plugin (@mzfr)
- 2019-03-17 Shell emulation pipe upgrade (@nunonovais)
- 2019-03-14 Shell emulation environment variables improved (@nunonovais)
- 2019-03-14 SSH crypto parameters now configurable in config file (@msharma)
- 2019-03-13 Disable keyboard-interactive authentication by default with option to enable
- 2019-03-13 Added *wc*, *crontab*, *chpasswd* command (@nunonovais)
- 2019-
- 2019-03-07 Output of *ssh -V* now configurable in cowrie.cfg with ssh_version setting
- 2019-03-07 Multiple timezone support in cowrie.cfg timezone directive. Default timezone is now UTC for both cowrie.log and cowrie.json
- 2019-03-12 Handle multiple password prompt. Option to enable or disable keyboard interactive prompt.

6.14 Release 1.5.3

- 2019-01-27 Telnet NAWS negotation removed to stop NMAP cowrie detection
- 2019-01-27 Various fixes for Python2/3 compatibility
- 2019-01-09 Documentation converted to ReStructuredText
- 2018-12-04 Fixes for VT outut plugin to only submit new files

6.15 Release 1.5.2

- 2018-11-19 Fix tftp exception and tftp test
- 2018-11-14 Remove *dblog* mechanism and *splunk* legacy output plugin.
- 2018-11-01 Add Python3 support for Splunk output plugin
- 2018-10-23 Improved free command
- 2018-10-20 Improved uname command
- 2018-10-16 Save VT results to JSON log

6.16 Release 1.5.1

- 2018-10-13 Fixes VT uploads, tab completion on Python3, Hassh support, setuptools functional. userdb migration
- 2018-09-07 NOTE! data/userdb.txt has moved to etc/userdb.txt and a default config is no longer provided!
- 2018-08-25 Downloads and TTY logs have moved to the var/ directory
- 2018-08-11 SSH keys now stored in var/lib/cowrie

- 2018-07-21 source code has move to the src/ directory. Delete old directories twisted/cowrie with compiled code
- 2018-06-29 txtcmds have been moved to share/cowrie/txtcmds
- 2018-06-28 filesystem config entry has changed. please verify if you have custom entry or pickle file
- 2018-06-23 fingerprint log message now holds KEX attributes and a unique fingerprint for the client
- 2018-04-27 Output plugins now require the mandatory config entry 'enabled'.
- 2018-02-06 cowrie.log now uses same rotation mechanism as cowrie.json. One file per day, rather than the default 1MB per file.
- 2017-12-13 Default umask for logs is now 0007. This means group members can access.
- 2017-10-24 Can store uploaded and downloaded artifacts to S3
- 2017-09-23 First proxy implementation for exec commands only
- 2017-07-03 Cuckoo v2 integration
- 2017-05-16 now combines config files: cowrie.cfg.dist and cowrie.cfg in this order
- 2017-05-09 start.sh and stop.sh have been replace by bin/cowrie start|stop
- 2017-04-27 New syntax "listen_endpoints" for configuring listening IP addresses/portnumbers
- 2017-03-15 SSH Forwarding/SFTP/keys/version config have been moved to [ssh]. Change your config file!
- 2017-02-12 Implemented toggle for SSH forwarding
- 2016-08-22 Merged Telnet support by @obilodeau!
- 2016-08-20 Update your libraries! 'configparser' now required: "pip install configparser"
- 2016-05-06 Load pickle once at startup for improved speed
- 2016-04-28 files in utils/ have been moved to bin/
- 2016-01-19 Support openssh style delayed compression
- 2016-01-13 Correct '.' support and +s and +t bits in ls
- 2016-01-13 Full username/group in SFTP ls
- 2016-01-05 Basic VirusTotal support has been added
- 2016-01-04 No longer crash when client tries ecdsa
- 2015-12-28 Interact port (default 5123) only listens on loopback interface now (127.0.0.1)
- 2015-12-24 Redirect to file (>) now works for most commands and is logged in dl/ directory
- **2015-12-06 UID information is now retrieved from honeyfs/etc/passwd. If you added additional users** you will need to add these to the passwd file as well
- 2015-12-04 New 'free' command with '-h' and '-m' options
- 2015-12-03 New 'env' command that prints environment variables
- 2015-02-02 Now use honeyfs/etc/passwd and group to get uid/gid info
- 2015-11-29 Size limit now enforced for SFTP uploads
- 2015-11-25 New 'sudo' command added
- **2015-11-19 Queued input during commands is now sent to shell to be executed** when command is finished
- 2015-11-18 Added SANS DShield output (Thanks @UnrealAkama)

- 2015-11-17 Added ElasticSearch output (Thanks @UnrealAkama)
- 2015-11-17 Standard input is now saved with SHA256 checksum. Duplicate data is not saved
- 2015-11-12 New 'busybox' command added (Thanks @mak)
- **2015-09-26 keyboard-interactive is back as authentication method, after**
  Twisted removed support initially
- 2015-07-30 Local syslog output module
- 2015-06-15 Cowrie now has a '-c' startup switch to specify the configuration file
- 2015-06-15 Removed exec_enabled option. This feature is now always enabled
- 2015-06-03 Cowrie now uses twisted plugins and has gained the '-p' commandline option
- 2015-06-01 Cowrie no longer search for config files in /etc and /etc/cowrie
- 2015-04-12 JSON output is now default via 'output' plugin mechanism. Rotates daily
- 2015-04-10 Fix for downloading files via SFTP
- 2015-03-31 Small tweaks on session close, closing session does not close ssh transport
- 2015-03-18 Merged 'AuthRandom' login class by Honigbij
- **2015-02-25 Internals for dblog/ modules changed completely.**
  Now accepts structured logging arguments, and uses eventids instead of regex parsing
- 2015-02-20 Removed screen clear/reset on logout
- 2015-02-19 Configuration directives have changed! ssh_addr has become listen_addr and ssh_port has become listen_port. The old keywords are still accepted for backwards compatibility
- default behaviour is changed to disable the exit jail
- sftp support
- exec support
- **stdin is saved as a file in dl/ when using exec commands**
  to support commands like 'cat >file; ./file'
- allow wget download over non-80 port
- simple JSON logging added
- accept log and deny publickey authentication
- add uname -r, -m flags
- add working sleep command
- enabled ssh diffie-hellman-group-exchange-sha1 algorithm
- add 'bash -c' support (no effect option)
- enable support for && multiple commands
- create uuid to uniquely identify each session
- log and deny direct-tcpip attempts
- add "chattr" command
- support emacs keybindings (c-a, c-b, c-f, c-p, c-n, c-e)
- add "sync" command

- accept, log and deny public key authentication
- add “uname -r” support
- logstash and kibana config files added, based on JSON log
- fix for honeypot detection (pre-auth differences with openssh)
- added verbose logging of client requested key exchange parameters (for client fingerprinting)
- fixes for behavior with non-existent files (cd /test, cat /test/nonexistent, etc)
- fix for ability to ping/ssh non-existent IP address
- always send ssh exit-status 0 on exec and shell
- ls output is now alphabetically sorted
- banner_file is deprecated. honeyfs/etc/issue.net is default
- add ‘dir’ alias for ‘ls’
- add ‘help’ bash builtin
- add ‘users’ aliased to ‘whoami’
- add ‘killall’ and ‘killall5’ aliased to nop
- add ‘poweroff’ ‘halt’ and ‘reboot’ aliases for shutdown
- add environment passing to commands
- added ‘which’, ‘netstat’ and ‘gcc’ from kippo-extra
- logging framework allows for keyword use

CHAPTER

SEVEN

USING THE PROXY

The SSH and Telnet proxies can be used to provide a fully-fledged environment, in contrast to the emulated shell traditionally provided by Cowrie. With a real backend environment where attackers can execute any Unix command, Cowrie becomes a high-interaction honeypot.

To use the proxy, start by changing the `backend` option to `proxy` in the `[honeypot]` section. In the remainder of this guide we will refer to the `[proxy]` section of the config file.

7.1 Choosing a Backend

Cowrie supports a simple backend (i.e., a real machine or virtual machines provided by you), but you can use Cowrie's backend pool, which provides a set of VMs, handling their boot and cleanup, also ensuring that different attackers (different IPs) each see a "fresh" environment, while connections from the same IP get the same VM.

VERY IMPORTANT NOTE: some attacks consist of downloading malicious software or accessing illegal content through insecure machines (such as your honeypot). If you are using your **own backend**, be sure to restrict networking to the Internet on your backend, and ensure other machines on your local network are isolated from the backend machine. The backend pool restricts networking and does its best to ensure total isolation, to the best of Qemu/libvirt (and our own) capabilities. **Be very careful to protect your network and devices!**

7.2 Configuring the Proxy

7.2.1 Backend configs

If you choose the simple backend, configure the hosts and ports for your backend. For the backend pool, configure the variables starting with `pool\_`. You'll also need to deal with the `[backend_pool]` section, which we detail in the Backend Pool's own documentation.

The backend pool can be run in the same machine as Cowrie, or on a remote one (e.g. Cowrie on a Raspberry Pi, and the pool in a larger machine). In the former case, set `pool` to `local`; in the later, set `pool` to `remote` and specify its host and port, matching with the `listen_endpoints` of the `[backend_pool]` section. Further configurations sent by the client are explained in Backend Pool's own documentation.

7.2.2 Authentication

Regardless of the used type of backend, Cowrie will need credentials to access the machine. These can be of any account on it, as long as it supports password authentication.

Note that these are totally independent of the credentials attackers can use (as set in `userdb`). `userdb` credentials are the ones attackers may use to connect to Cowrie, while `backend_user` and `backend_pass` are used to connect Cowrie to the backend.

7.2.3 Telnet prompt detection

Due to the different implementations of Telnet, there is not a single reliable way of catching the authentication phase of the protocol as in SSH. Therefore, we rely on regex expressions to detect authentication prompts, allowing us to identify the credentials supplied by the attacker and check if they are accepted by `userdb`. If they are, we send the `backend_user` and `backend_pass` to the backend (spoofing the authentication); if not, we send `backend_pass` appended with the word `fake` to force a login failed prompt (and fail authentication overall).

If you don't want to spoof authentication, set `telnet_spoof_authentication` to false. In this mode, only the backend real details will be accepted to authenticate, thus bypassing `userdb`.

The expressions to detect authentication prompts are `telnet_username_prompt_regex` and `telnet_password_prompt_regex`. A further expression we use is defined in `telnet_username_in_negotiation_regex`. Some clients send their username in the first phases of the protocol negotiation, which some systems (the backend) use to only show the password prompt the first time authentication is tried (thus assuming the client's username as the username they'll use to login into the system). Cowrie tries to capture this username and use it when comparing the auth details with the `userdb`.

7.2.4 Analysing traffic

Analysing raw traffic can be interesting when setting up Cowrie, in particular to set-up Telnet prompt detection. For this, you can set `log_raw` to true.

CHAPTER

EIGHT

USING THE LLM BACKEND

The LLM (Large Language Model) backend uses AI models like OpenAI's GPT to generate realistic shell responses. Instead of static command emulation, the LLM dynamically generates output for any command, making the honeypot more convincing and capable of handling unexpected inputs.

This is an experimental feature that provides a high-interaction honeypot experience without requiring a real backend system.

8.1 Enabling the LLM Backend

To use the LLM backend, change the `backend` option to `llm` in the `[honeypot]` section:

```
[honeypot]
backend = llm
```

Then configure the `[llm]` section with your API credentials.

8.2 Configuration

8.2.1 API Key (Required)

You must provide an API key for the LLM service. For OpenAI:

```
[llm]
api_key = sk-your-api-key-here
```

Get your API key from https://platform.openai.com/api-keys

8.2.2 Model Selection

Choose which model to use. Smaller models are faster and cheaper, larger models may provide more realistic responses:

```
[llm]
model = gpt-4o-mini
```

Common options:

- `gpt-4o-mini` - Fast and cost-effective (default)
- `gpt-4o` - More capable, higher cost
- `gpt-4-turbo` - High capability

8.2.3 Custom API Endpoints

To use a different OpenAI-compatible API (such as a local LLM server), configure the host and path:

```
[llm]
host = https://api.openai.com
path = /v1/chat/completions
```

For local LLM servers like Ollama or text-generation-webui, point to your local endpoint:

```
[llm]
host = http://localhost:11434
path = /v1/chat/completions
```

8.2.4 Response Parameters

Control how the LLM generates responses:

```
[llm]
# Maximum tokens in the response (default: 500)
max_tokens = 500

# Temperature: 0.0-2.0, higher = more random (default: 0.7)
temperature = 0.7
```

8.2.5 Debugging

Enable debug logging to see LLM requests and responses:

```
[llm]
debug = true
```

This logs the full request/response JSON to the Cowrie log file.

8.3 How It Works

When an attacker connects and enters a command:

1. The command is sent to the LLM along with a system prompt that instructs it to simulate a Linux server
2. The LLM generates realistic command output
3. The response is displayed to the attacker
4. Command history is maintained to provide context for follow-up commands

The LLM maintains conversation history (last 10 commands) to provide consistent responses across a session. For example, if the attacker runs `cd /tmp` followed by `pwd`, the LLM will correctly respond with `/tmp`.

8.4 Advantages

- **No static signatures**: Every response is dynamically generated
- **Handles any command**: Unlike the shell backend, unknown commands get realistic responses
- **Consistent sessions**: Maintains context across commands

- **Easy setup**: No virtual filesystem or backend VMs required

8.5 Limitations

- **API costs**: Each command requires an API call
- **Latency**: Responses take 1-3 seconds depending on the model
- **State consistency**: The LLM may occasionally be inconsistent with filesystem state
- **No real execution**: Downloads and file operations are simulated, not real

8.6 Security Considerations

- Your API key is sent with every request - keep your configuration file secure
- LLM responses are logged - review logs for any unexpected content
- The LLM cannot execute real commands - all responses are text-only

8.7 Example Configuration

Minimal configuration:

```
[honeypot]
backend = llm

[llm]
api_key = sk-your-api-key-here
```

Full configuration with all options:

```
[honeypot]
backend = llm

[llm]
api_key = sk-your-api-key-here
model = gpt-4o-mini
host = https://api.openai.com
path = /v1/chat/completions
max_tokens = 500
temperature = 0.7
debug = false
```

CHAPTER

NINE

CHANGING THE COWRIE FILE SYSTEM

9.1 Introduction

Part of Cowrie is an emulated file system. Each honeypot visitor will get their own personal copy of this file system and this will be deleted when they log off. They can delete or change any file, nothing will be preserved.

The file system implementation consists of two parts: the *pickle* file, which mostly holds metadata for the files (filename, directory, permissions, owner, size, file type, etc) but has contents for a few files. Most files have no content.

The *honeyfs* directory holds user created file contents, this overrides content from the pickle file and is a quick way to have custom content

To show the contents of the file, it needs both a meta data entry (pickle) and a honeyfs file.

9.2 Creating a new pickle file

Create a directory where you put all files you'd like to be show in your filesystem Create the pickle file:

```
$ source cowrie-env/bin/activate
(cowrie-env) $ createfs -l YOUR-DIR -d DEPTH -o custom.pickle
```

Make sure your config picks up custom.pickle, by referencing it in *cowrie.cfg*:

```
[shell]
filesystem = custom.pickle
```

Or set an environment variable:

```
$ export COWRIE_SHELL_FILESYSTEM=custom.pickle
```

9.3 Customizing text command output

Some commands in Cowrie are implemented as simple text output files under `txtcmds`. Operators can point Cowrie at a custom directory with `[honeypot] txtcmds_path`:

```
[honeypot]
txtcmds_path = /opt/cowrie/txtcmds
```

The command path below that directory must match the path in the virtual filesystem. For example, to customize `/usr/bin/lscpu` output, create:

```
/opt/cowrie/txtcmds/usr/bin/lscpu
```

The command still needs an entry in the virtual filesystem pickle, the same way files in `honeyfs` need matching metadata. If a command is not present under `txtcmds_path`, Cowrie falls back to the bundled `cowrie.data/txtcmds` output.

CHAPTER

TEN

BACKEND POOL

The Backend Pool manages a set of dynamic backend virtual machines to be used by Cowrie's proxy. The pool keeps a set of VMs running at all times, ensuring different attackers each see a "pristine" VM, while repeated connections from the same IP are served with the same VM, thus ensuring a consistent view to the attacker. Furthermore, VMs in the pool have their networking capabilities restricted by default: some attacks consist of downloading malicious software or accessing illegal content through insecure machines (such as your honeypot). Therefore, we limit any access to the Internet via a network filter, which you can configure as you see fit.

The VMs in the backend pool, and all infrastructure (snapshots, networking and filtering) are backed-up by Qemu/libvirt. We provide two example VM images (for Ubuntu Server 18.04 and OpenWRT 18.06.4) whose configurations are already set and ready to be deployed. Further below in this guide we'll discuss how to create your own images and customise libvirt's XML configuration files.

First of all, install the needed dependencies for the pool, as explained in the installation steps.

10.1 Authorization

Add your cowrie user to the `libvirt` group to ensure you have permission to run the VMs on the backend server

```
sudo usermod -aG libvirt "COWRIE_USER_HERE"
```

10.2 Provided images

To allow for a simple setup, we provide two VM images to use with the backend pool: Ubuntu 18.04 and OpenWRT. You can download them below, and then edit `cowrie.cfg`'s `guest_image_path` to match the path of the images. In the case of OpenWRT you will need two different files. Note that a separate set of configs is provided for each image in the default configuration. Choose the one you want to use and comment the other as needed.

- Ubuntu 18.04.
- OpenWRT disk image.
- OpenWRT kernel image.

10.3 Backend Pool initialisation

Depending on the machine that will be running the backend pool, initialisation times for VMs can vary greatly. If the pool is correctly configured, you will get the *PoolServerFactory starting on 6415* message on your log.

After a while, VMs will start to boot and, when ready to be used, a message of the form *Guest 0 ready for connections @ 192.168.150.68! (boot 17s)* will appear for each VM. Before VMs are ready, SSH and Telnet connections from attackers will be dropped by Cowrie.

10.4 Proxy configurations

When the proxy starts, and regardless whether the backend pool runs on the same machine as the proxy or not, some configurations are sent by the proxy to the pool during runtime.

These are:

- **pool_max_vms**: the number of VMs to be kept running in the pool
- **pool_vm_unused_timeout**: how much time (seconds) a used VM is kept running (so that an attacker that reconnects is served the same VM.
- **apool_share_guests**: what to do if no "pristine" VMs are available (i.e., all have been connected to); if set to true we serve a random one from the used, if false we throw an exception.

10.5 Backend Pool configuration

In this section we'll discuss the `[backend_pool]` section of the configuration file.

The backend pool can be run in the same machine as the rest of Cowrie, or in a separate one. In the former case, you'd be running Cowrie with

```
[backend_pool]
pool_only = false

[proxy]
backend = pool
pool = local
```

If you want to deploy the backend pool in a different machine, then you'll need to invert the configuration: the pool machine has `pool_only = true` (SSH and Telnet are disabled), and the proxy machine has `pool = remote`.

Note: The communication protocol used between the proxy and the backend pool is unencrypted. Although no sensitive data should be passed, we recommend you to only use private or local interfaces for listening when setting up `listen_endpoints`.

10.5.1 Recycling VMs

Currently, handling of virtual machines by the pool is not perfect. Sometimes, VMs reach an inconsistent state or become unreliable. To counter that, and ensure fresh VMs are ready constantly, we use the `recycle_period` to periodically terminate running instances, and boot new ones.

10.5.2 Snapshots

VMs running in the pool are based on a base image that is kept unchanged. When booting, each VM creates a snaphost that keeps track of differences between the base image and snapshot. If you want to analyse snapshots and see any changes made in the VMs, set `save_snapshots` to true. If set to true be mindful of space concerns, each new VM will take at least ~20MB in storage.

10.5.3 XML configs (advanced)

The default libvirt XML templates (`default_guest.xml`, `default_filter.xml`, `default_network.xml`, plus the architecture- specific `aarch64_guest.xml`, `wrt_arm_guest.xml`, `wrt_x86_guest.xml`) ship inside the cowrie package at `src/cowrie/data/pool_configs/`. If you are using one of the default images, you most likely do not need to change them.

To override one or more of these templates without touching the bundled copies, set `[backend_pool] config_files_path` in `cowrie.cfg` to a directory of your choice and drop the files you want to replace into it. Cowrie reads each XML by basename:

1. `<config_files_path>/<basename>.xml` if the file exists; else
2. the bundled `src/cowrie/data/pool_configs/<basename>.xml`.

Only the files you actually override need to exist in your directory; anything missing falls through to the bundled copy. Leave `config_files_path` unset to use the bundled templates for all of them.

10.5.4 Guest configurations

A set of guest (VM) parameters can be defined as we explain below:

- **guest_config**: the XML configuration for the guest (default_guest.xml works for x86 machines, and wrt_arm_guest.xml for ARM-based OpenWRT)
- **guest_privkey**: currently unused
- **guest_tag**: an identifiable name for snapshots and logging
- **guest_ssh_port / guest_telnet_port**: which ports are listening for these on the VM (no relation with the ports Cowrie's listening to)
- **guest_image_path**: the base image upon which all VMs are created from
- **guest_hypervisor**: the hypervisor used; if you have an older machine or the emulated architecture is different from the host one, then use software-based "QEMU"; however, if you are able to, use "KVM", it's **much** faster.
- **guest_memory**: memory assigned to the guest; choose a value considering the number of guests you'll have running in total (`pool_max_vms`)

10.5.5 NATing

VMs are assigned an IP in a local network defined by libvirt. If you need to access the VMs from a different machine (i.e., running the backend pool remotely), then an external-facing IP (as defined in `nat_public_ip`) is needed for the proxy to connect to.

For this purpose, we provide a simple form of NAT that, for each VM request, and if enabled, starts a TCP proxy to forward data from a publicly-acessible IP to the internal libvirt interface.

10.6 Creating VM images

Creating a new type of VM involves three steps: creating a base image, installing the OS, and tweaking configs.

To create a disk image issue

```
sudo qemu-img create -f qcow2 image-name.qcow2 8G
```

(the qcow2 format is needed to ensure create snapshots, thus providing isolation between each VM instance; you can specify the size you want for the disk)

Then you'll have to install an OS into it

```
virt-install --name temp-domain --memory 1024 --disk image-name.qcow2 --cdrom os-install-
↪cd.iso --boot cdrom
```

(to use virt-install you need to install the virtinst package)

After install check that the VM has network connectivity. If you set the pool to use the created image and SSH does not come up, log into the VM via libvirt (e.g., using virt-manager) and try the following (might change depending on system)

```
# run all as root
ip link show
ip link set enp1s0 up
dhclient
```

In Ubuntu dhclient can be set to run with netplan, for example, to be run on startup.

10.6.1 Set up Telnet

Steps used in Ubuntu, can be useful in other distros

```
# run all as root
apt-get -y install telnetd xinetd
touch /etc/xinetd.d/telnet
printf "service telnet\n{\ndisable = no\nflags = REUSE\nsocket_type = stream\nwait = no\
↪nuser = root\nserver = /usr/sbin/in.telnetd\nlog_on_failure += USERID\n}" > /etc/
↪xinetd.d/telnet
printf "pts/0\npts/1\npts/2\npts/3\npts/4\npts/5\npts/6\npts/7\npts/8\npts/9" >> /etc/
↪securetty
service xinetd start
```

10.6.2 Customising XML configs

If you want, you can customise libvirt's XML configurations.

The main configuration for a guest is located in `default_guest.xml`. This defines the virtual CPU, available memory, and devices available on the guest. Most of these configurations are set by Cowrie using the guest configurations; you'll see them in the XML as templates ("{guest_name}"). The main blocks of XML regard the disk and network interface devices.

You can include these blocks as-is in your custom configuration to allow Cowrie to manage your custom guests automatically.

```
<disk type='file' device='disk'>
    <driver name='qemu' type='qcow2'/>
    <source file='{disk_image}'/>
    <target dev='vda' bus='virtio'/>
    <address type='pci' domain='0x0000' bus='0x03' slot='0x00' function='0x0'/>
</disk>
```

```
<interface type='network'>
    <start mode='onboot'/>
    <mac address='{mac_address}'/>
    <source network='{network_name}'/>
    <model type='virtio'/>
    <address type='pci' domain='0x0000' bus='0x01' slot='0x00' function='0x0'/>
    <filterref filter='cowrie-default-filter'/>
</interface>
```

The other important configuration file is `default_filter.xml`, which handles how networking is restricted in the guest VM (aka to the attackers). This file is composed by a set of rules of the form

```
<rule action='accept' direction='in'>
    <tcp dstportstart='22'/>
</rule>
```

Each rule specifies a type of traffic (TCP, UDP…) and direction, whether to accept or drop that traffic, and the destiantion of traffic. The default filter provided allows inbound SSH and Telnet connections (without which the VM would be unusable, outbound ICMP traffic (to allow pinging) and outbound DNS querying. All other traffic is dropped as per the last rule, thus forbidding any download or tunnelling.

VERY IMPORTANT NOTE: some attacks consist of downloading malicious software or accessing illegal content through insecure machines (such as your honeypot). Our provided filter restricts networking and does its best to ensure total isolation, to the best of Qemu/libvirt (and our own) capabilities. **Be very careful to protect your network and devices while allowing any more traffic!**

10.7 References

- libvirt guest XML syntax
- libvirt network filter XML syntax
- Create a OpenWRT image

CHAPTER

ELEVEN

ANALYSING SNAPSHOTS AND DOWNLOADED CONTENT

One interesting aspect of Cowrie is the capability to analyse any downloaded malware and content into the honeypot. The snapshot mechanism can be leveraged to analyse any download and any change performed against the base image, to determine which files have been changed.

This guide shows how that can be achieved by leveraging using the `libguestfs-tools` package.

11.1 Getting the a filesystem diff

The first step is getting the differences between each VM that was used and the base image provided. The tool we'll be using is `virt-diff`, which provides a similar syntax to that of Unix's `diff`.

```
$ sudo virt-diff -a ~/cowrie-imgs/ubuntu18.04-minimal.qcow2 -A snapshot-ubuntu18.04-
↪a70b9671ad4d44619af2c4a41a28aec0.qcow2
```

(the tool might need to be run with sudo due to a permission denied error)

The output will contain all changed files and their content, which might get long easily. The following command outputs the names of changed files, to be easier to read (assuming the output from `virt-diff` is stored in a file diff.txt)

```
$ grep -aE "^\+ |^- |^= " diff.txt
```

Here is an example output, in a VM were we created a file called `avirus`:

```
= - 0644       1024 /boot/grub/grubenv
= - 0600       1036 /root/.bash_history
+ - 0644         14 /root/avirus
+ d 0700       4096 /tmp/systemd-private-9f5f6c41f75f48f4991c55f3fc3d6435-systemd-
↪resolved.service-syUmHS
+ d 1777       4096 /tmp/systemd-private-9f5f6c41f75f48f4991c55f3fc3d6435-systemd-
↪resolved.service-syUmHS/tmp
+ d 0700       4096 /tmp/systemd-private-9f5f6c41f75f48f4991c55f3fc3d6435-systemd-
↪timesyncd.service-SrDysr
+ d 1777       4096 /tmp/systemd-private-9f5f6c41f75f48f4991c55f3fc3d6435-systemd-
↪timesyncd.service-SrDysr/tmp
+ - 0644       2163 /var/backups/apt.extended_states.0
+ - 0644          0 /var/lib/apt/daily_lock
= - 0644          0 /var/lib/private/systemd/timesync/clock
= - 0600        512 /var/lib/systemd/random-seed
= - 0644          0 /var/lib/systemd/timers/stamp-apt-daily-upgrade.timer
= - 0644          0 /var/lib/systemd/timers/stamp-apt-daily.timer
= - 0644          0 /var/lib/systemd/timers/stamp-fstrim.timer
```

(continues on next page)

(continued from previous page)

```
= - 0644          0 /var/lib/ubuntu-release-upgrader/release-upgrade-available
= - 0640      14534 /var/log/auth.log
= - 0640    8388608 /var/log/journal/19497399992e49388d57aa395b993b2c/system.journal
= - 0640     346383 /var/log/kern.log
= - 0664     292292 /var/log/lastlog
= - 0640     436458 /var/log/syslog
= - 0664      19968 /var/log/wtmp
+ d 0700       4096 /var/tmp/systemd-private-9f5f6c41f75f48f4991c55f3fc3d6435-systemd-
↪resolved.service-u5dZk6
+ d 1777       4096 /var/tmp/systemd-private-9f5f6c41f75f48f4991c55f3fc3d6435-systemd-
↪resolved.service-u5dZk6/tmp
+ d 0700       4096 /var/tmp/systemd-private-9f5f6c41f75f48f4991c55f3fc3d6435-systemd-
↪timesyncd.service-Tcil4E
+ d 1777       4096 /var/tmp/systemd-private-9f5f6c41f75f48f4991c55f3fc3d6435-systemd-
↪timesyncd.service-Tcil4E/tmp
```

As you can see, the created file is shown among lots of log and temporary files. There is no good way to eliminate these, but we can use grep to ignore them:

```
$ grep -aE "^\+ |^- |^= " diff.txt | grep -aEv "/tmp/systemd|/var/log|/var/lib"
```

Which now gives us a clearer output:

```
= - 0644       1024 /boot/grub/grubenv
= - 0600       1036 /root/.bash_history
+ - 0644         14 /root/avirus
+ - 0644       2163 /var/backups/apt.extended_states.0
```

11.2 Getting interesting files

To be able to get and read the files you're interested in, you'll need to mount the snapshot into your machine and copy the file(s) into your disk. The steps we describe are taken from here, and rewritten here for clarity.

We start by mounting the image in a temporary dir:

```
$ mkdir /tmp/mount_qcow2
$ sudo guestmount -a snapshot-ubuntu18.04-a70b9671ad4d44619af2c4a41a28aec0.qcow2 -m /dev/
↪sda1 --ro /tmp/mount_qcow2
```

If we now search for the file in the mount directory we can see its contents, and then unmount the drive:

```
$ sudo ls -halt /tmp/mount_qcow2/root
total 32K
-rw-------  1 root root 1.1K Jul 28 21:45 .bash_history
drwx------  3 root root 4.0K Jul 28 21:45 .
-rw-r--r--  1 root root   14 Jul 28 21:45 avirus
drwxr-xr-x 22 root root 4.0K Jul 15 01:57 ..
-rw-r--r--  1 root root   74 Jul 15 00:59 .selected_editor
drwx------  2 root root 4.0K Jul 15 00:59 .cache
-rw-r--r--  1 root root 3.1K Apr  9  2018 .bashrc
-rw-r--r--  1 root root  148 Aug 17  2015 .profile
```

(continues on next page)

(continued from previous page)

```
$ sudo cat /tmp/mount_qcow2/root/avirus
virus content

$ sudo guestunmount /tmp/mount_qcow2/
```

Note: the device to be mounted from the image isn't always `/dev/sda1`. However, if you run the command as-is, `guestmount` will check if `/dev/sda1` exists and, if not, it will list available partitions for you.

CHAPTER

TWELVE

OUTPUT EVENT CODE REFERENCE

This guide documents the event id's used by Cowrie that are sent to the output modules, such as the JSON logging module.

12.1 Reference

12.1.1 Shared Attributes

These attributes are shared by all messages.

Attributes:

- *message*: human readable message
- *sensor*: name of the sensor, by default the hostname
- *timestamp*: timestamp in ISO8601 format in UTC time zone
- *src_ip*: attacker IP address
- *session*: unique session identifier

12.1.2 cowrie.client.fingerprint

If the attacker attemps to log in with an SSH public key this is logged here

Attributes:

- *username*: username
- *fingerprint*: the key fingerprint
- *key*: the key
- *type*: type of key, typically ssh-rsa or ssh-dsa

12.1.3 cowrie.login.success

Successful authentication.

Attributes:

- username
- password

12.1.4 cowrie.login.failed

Failed authentication.

Attributes:

- username
- password

12.1.5 cowrie.client.size

Width and height of the users terminal as communicated through the SSH protocol.

Attributes:

- width
- height

12.1.6 cowrie.session.file_upload

File uploaded to Cowrie, generally through SFTP or SCP or another way.

Attributes:

- filename
- outfile
- shasum

12.1.7 cowrie.session.file_download

File downloaded to Cowrie

Attributes:

- url
- outfile
- shasum

12.1.8 cowrie.command.input

Command line input

Attributes:

- input

12.1.9 cowrie.command.failed

Command line input failed

Attributes:

- input

12.1.10 cowrie.virustotal.scanfile

File sent to VT for scanning

Attributes:

- sha256
- is_new
- positives
- total

12.1.11 cowrie.session.connect

New connection

Attributes:

- src_ip
- src_port
- dst_ip
- dst_port

12.1.12 cowrie.client.version

SSH identification string

Attributes:

- version

12.1.13 cowrie.client.kex

SSH Key Exchange Attributes

Attributes:

- hassh
- hasshAlgorithms
- kexAlgs
- keyAlgs

12.1.14 cowrie.session.closed

Session closed

Attributes:

- *duration_ms*: duration of session in milliseconds

12.1.15 cowrie.session.params

Session parameters

Attributes:

- arch

12.1.16 cowrie.log.closed

TTY Log closed

Attributes:

- *duration_ms*: duration of session in milliseconds
- *ttylog*: filename of session log that can be replayed with `bin/playlog`
- *size*: size in bytes
- *shasum*: SHA256 checksum of the attacker input only (honeypot generated output is not included)
- *duplicate*: whether this is the first time this attack has been seen

12.1.17 cowrie.direct-tcpip.request

Request for proxying via the honeypot

Attributes:

- dst_ip
- dst_port
- src_ip
- src_port

12.1.18 cowrie.direct-tcpip.data

Data attempted to be sent through direct-tcpip forwarding

Attributes:

- dst_ip
- dst_port

12.1.19 cowrie.client.var

Attributes:

- name
- value

CHAPTER

THIRTEEN

HOW TO SEND COWRIE OUTPUT TO DATADOG LOG MANAGEMENT

This guide describes how to configure and send cowrie outputs to Datadog Log Management.

13.1 Datadog output module Prerequisites

- Working Cowrie installation
- Existing Datadog account.

13.2 Cowrie Configuration for Datadog output module

- Modify `cowrie.cfg` to enable the `[output_datadog]` section.
- Add an API Key. You may generate one for your organisation from here.
- Optionally customize `ddsource`, `ddtags` and `service`. Otherwise, defaults are respectively `cowrie`, `env:prod` and `honeypot`.

13.3 Datadog Configuration

JSON logs are handled without further configuration in Datadog.

CHAPTER

FOURTEEN

DOCKER REPOSITORY

Docker Images are available on Docker Hub: https://hub.docker.com/r/cowrie/cowrie

14.1 Docker Quick Start

To run Cowrie in Docker:

```
$ docker run -p 2222:2222 cowrie/cowrie
$ ssh -p 2222 root@localhost
```

14.2 Configuring Cowrie in Docker with Environment Variables

Cowrie in Docker can be configured using environment variables. The variable starts with `COWRIE_` then has the section name in capitals, followed by the stanza in capitals. This example enables telnet support:

```
COWRIE_TELNET_ENABLED=yes
```

And then start Cowrie as:

```
$ docker run -e COWRIE_TELNET_ENABLED=yes -p 2223:2223 cowrie/cowrie
$ telnet localhost 2223
```

14.3 Configuring Cowrie in Docker with Config Files

Alternatively, Cowrie in Docker can use an *etc* mount to store configuration data. Docker can either mount a volume or a directory.

Mounting a volume or directory on */etc* will make existing files unavailable to Cowrie, so make sure to copy *userdb.txt* and *cowrie.cfg.dist* there too!

Create `` `cowrie.cfg` `` inside the etc directory (or volume) with the following contents to enable telnet in Cowrie in Docker:

```
[telnet]
enabled = yes
```

Start Cowrie as:

```
$ docker run -p 2223:2223 --mount type=bind,source=./etc,target=/cowrie/cowrie-git/etc␣
↪cowrie/cowrie
$ telnet localhost 2223
```

Environment variables take precedence over the configuration file.

14.4 Building Docker Images

If you want to make extensive changes to the Docker image, it may be easier to build your own local docker image with:

```
$ make docker-load
```

CHAPTER

FIFTEEN

HOW TO SEND COWRIE OUTPUT TO AN ELK STACK

15.1 ElasticSearch Prerequisites

- Working Cowrie installation
- Cowrie JSON log file (enable `output_json` in `cowrie.cfg`)
- Java 8

15.2 ElasticSearch Installation

This is a simple setup for ELK stack, to be done on the same machine that is used for cowrie. We use *Filebeat* to send logs to *Logstash*, and we use *Nginx* as a reverse proxy to access *Kibana*. Note there are many other possible configurations!

Add Elastic's repository and key:

```
$ wget -qO - https://packages.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
$ echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" | sudo tee /etc/
↪apt/sources.list.d/elastic-7.x.list
$ apt-get update
```

Install logstash, elasticsearch, kibana and filebeat:

```
$ sudo apt -y install apt-transport-https wget default-jre
$ sudo apt install elasticsearch logstash kibana
$ sudo apt install filebeat
$ sudo apt install nginx apache2-utils
```

Enable the services:

```
$ sudo systemctl enable elasticsearch logstash kibana filebeat nginx
```

15.3 ElasticSearch Configuration

ElasticSearch configuration file is located in `/etc/elasticsearch/elasticsearch.yml`. The default settings need not be changed.

If you are only operating a single ElasticSearch node, you can add the following configuration item:

```
discovery.type: single-node
```

By default, ElasticSearch listens on port 9200. Test it:

```
curl http://localhost:9200
```

You should get a JSON object in return.

15.4 Kibana Configuration

Make a folder for logs:

```
$ sudo mkdir /var/log/kibana
$ sudo chown kibana:kibana /var/log/kibana
```

Change the following parameters in `/etc/kibana/kibana.yml` to reflect your server setup:

- `server.host` - set it to *localhost* if you use nginx for basic authentication or external interface if you use XPack (see below)
- `server.name` - name of the server
- `elasticsearch.hosts` - address of the elasticsearch: ["http://localhost:9200"]
- `elasticsearch.username` - only needed only if you use XPack (see below)
- `elasticsearch.password` - only needed only if you use XPack (see below)
- `logging.dest` - set path to logs (*/var/log/kibana/kibana.log*)

15.5 Logstash Configuration

Get GeoIP data from www.maxmind.com (free but requires registration): download the GeoLite2 City GZIP. Unzip it and locate the mmdb file. Place it somewhere in your filesystem and make sure that "logstash" user can read it:

```
$ sudo mkdir -p /opt/logstash/vendor/geoip/
$ sudo mv GeoLite2-City.mmdb /opt/logstash/vendor/geoip
```

Configure logstash:

```
$ sudo cp logstash-cowrie.conf /etc/logstash/conf.d
```

Make sure the configuration file is correct. Check the input section (path), filter (geoip databases) and output (elasticsearch hostname):

```
$ sudo systemctl restart logstash
```

15.6 FileBeat Configuration

FileBeat is not mandatory (it is possible to directly read Cowrie logs from Logstash) but nice to have, because if Logstash is under pressure, it automatically knows to slow down + it is possible to deal with multiple sensor inputs.

Configure filebeat:

```
$ sudo cp filebeat-cowrie.conf /etc/filebeat/filebeat.yml
```

Check the following parameters:

```
filebeat.inputs: the path must point to cowrie's json logs
output.elasticsearch: must be false because we want Filebeat to send to Logstash, not␣
↪directly to ElasticSearch
output.logstash: must be true. The default port for Logstash is 5044, so hosts should be␣
↪["localhost:5044"]
```

Start filebeat:

```
$ sudo systemctl start filebeat
```

15.7 Nginx

ELK has been configured on localhost. If you wish to access it remotely, you can setup a reverse proxy to Kibana's backend server, which runs on port 5601 by default.

Install Nginx:

```
$ sudo apt install nginx apache2-utils
```

Create an administrative Kibana user and password:

```
$ sudo htpasswd -c /etc/nginx/htpasswd.users admin_kibana
```

Edit Nginx configuration /etc/nginx/sites-available/default. Customize port to what you like, and specify your server's name (or IP address):

```
server {
     listen YOURPORT;

     server_name YOURIPADDRESS;

     auth_basic "Restricted Access";
     auth_basic_user_file /etc/nginx/htpasswd.users;

     location / {
           proxy_pass http://localhost:5601;
           proxy_http_version 1.1;
           proxy_set_header Upgrade $http_upgrade;
           proxy_set_header Connection 'upgrade';
           proxy_set_header Host $host;
           proxy_cache_bypass $http_upgrade;
     }
}
```

Start the service:

```
$ sudo systemctl start nginx
```

15.8 Using Kibana

You can list indexes with:

```
$ curl 'http://localhost:9200/_cat/indices?v'
```

You should see a Cowrie index cowrie-logstash-DATE... Its health is yellow because the number of replicas should be set to 0 (unless you want another configuration):

```
$ curl -XPUT 'localhost:9200/cowrie-logstash-REPLACEHERE/_settings' -H "Content-Type:␣
↪application/json" -d '{ "index" : {"number_of_replicas" : 0 } }'
```

It should answer {"acknowledged":true}

In Kibana's GUI, create an index pattern (Management / Index Patterns) for

```
cowrie-logstash-*
```

Use default settings and timestamp.

15.9 Tuning ELK stack

Refer to Elastic's documentation about proper configuration of the system for the best ElasticSearch's performance

You may avoid installing nginx for restricting access to Kibana by installing official Elastic's plugin called "X-Pack" (https://www.elastic.co/products/stack)

ELK log files get big: ensure you have enough space in /var, consider setting up LVM or ZFS partitions.

15.10 ElasticSearch Troubleshooting

- View service logs with: `sudo journalctl -u service`
- If the date in Kibana is incorrect, check (Advanced Settings / dateFormat)

CHAPTER

SIXTEEN

HOW TO SEND COWRIE OUTPUT TO GRAYLOG

This guide describes how to configure send cowrie outputs to graylog via syslog and http gelf input.

16.1 Prerequisites

- Working Cowrie installation
- Working Graylog installation

16.2 Cowrie Configuration

16.2.1 Using Syslog

Open the Cowrie configuration file and enable localsyslog output:

```
[output_localsyslog]
enabled = true
facility = USER
format = text
```

Restart Cowrie

16.2.2 Using GELF HTTP Input

Open the Cowrie configuration file and find this block

```
[output_graylog]
enabled = false
url = http://127.0.0.1:12201/gelf
```

Enable this block and specify url of your input.

Restart Cowrie

16.3 Graylog Configuration

16.3.1 Syslog Input

Open the Graylog web interface and click on the **System** drop-down in the top menu. From the drop-down menu select **Inputs**. Select **Syslog UDP** from the drop-down menu and click the **Launch new input** button. In the modal dialog enter the following information:

```
**Title:** Cowrie
**Port:** 8514
**Bind address:** 127.0.0.1
```

Then click **Launch.**

16.3.2 GELF HTTP Input

Open the Graylog web interface and click on the **System** drop-down in the top menu. From the drop-down menu select **Inputs**. Select **GELF HTTP** from the drop-down menu and click the **Launch new input** button. In the modal dialog enter the information about your input.

Then click **Launch.**

Note:

- Do not remove **/gelf** from the end of URL block, expect of case when your proxing this address behind nginx;

16.3.3 Parsing Cowrie JSON

Extractor

Click **Manage Extractors** near created input. On new page click **Actions** -> **Import extractors** and paste this config

```
{
  "extractors": [
    {
      "title": "Cowrie Json Parser",
      "extractor_type": "json",
      "converters": [],
      "order": 0,
      "cursor_strategy": "copy",
      "source_field": "message",
      "target_field": "",
      "extractor_config": {
        "list_separator": ", ",
        "kv_separator": "*",
        "key_prefix": "",
        "key_separator": "_",
        "replace_key_whitespace": false,
        "key_whitespace_replacement": "_"
      },
      "condition_type": "none",
      "condition_value": ""
    }
  ],
  "version": "4.2.1"
}
```

Pipeline

When running Graylog with the Forwarder input, traditional extractors are not available. Instead, you can use a pipeline rule to parse the JSON data.

Create a Stream and add the Cowrie logs to it.

Streams -> **Create Stream** -> **Title:** Cowrie -> **Description:** Cowrie logs -> **Create Stream**

Create a Stream Rule for the Cowrie Stream.

Streams -> **Cowrie** -> **Manage Rules** -> **Add Stream Rule** -> **Type:** *match input* **Input:** *Cowrie (GELF HTTP)* -> **Save**

Create a Pipeline Rule for the Cowrie Stream.

System -> **Pipelines** -> **Manage rules** -> **Create Rule** -> **Use Source Code Editor**

Paste the following code into the Rule source:

```
rule "Parse Cowrie message"
when
  has_field("message")
then
  // If you want to keep the original message, uncomment the following line and comment␣
↪out the next line.
  //let json_string = regex_replace("\"message\"", to_string($message.message), "\
↪"cowrie_message\"");
  let json_string = to_string($message.message);
  let json = parse_json(json_string);
  let map = to_map(json);
  set_fields(map);
end
```

Create a Pipeline for the Cowrie Stream.

System -> **Pipelines** -> **Manage pipelines** -> **Add new pipeline** -> **Title:** *Parse Cowrie logs* -> **Description:** Cowrie logs -> **Create Pipeline**

Under the **Pipeline connections** section, connect the Cowrie Stream to the Pipeline by clicking the **Edit connections** button and selecting the Cowrie Stream.

Under Pipeline Stages, edit Stage 0 and add the Pipeline Rule to the Stage.

16.4 Syslog Configuration (For Syslog Output only)

Create a rsyslog configuration file in /etc/rsyslog.d:

```
$ sudo nano /etc/rsyslog.d/85-graylog.conf
```

Add the following lines to the file:

```
$template GRAYLOGRFC5424,"<%pri%>%protocol-version% %timestamp:::date-rfc3339% %HOSTNAME
↪% %app-name% %procid% %msg%\n"
*.* @127.0.0.1:8514;GRAYLOGRFC5424
```

Restart rsyslog:

```
$ sudo service rsyslog restart
```

CHAPTER

SEVENTEEN

HOW TO SEND COWRIE OUTPUT TO KIPPO-GRAPH

17.1 Kippo-Graph Prerequisites

- Working Cowrie installation
- LAMP stack (Linux, Apache, MySQL, PHP)

17.2 Kippo-Graph Installation

This covers a simple installation, with kippo-graph and Cowrie on the same server. Please see here for installation: https://github.com/ikoniaris/kippo-graph

17.3 MySQL configuration for Kippo-Graph

Configuring Cowrie requires setting up the SQL tables and then telling Cowrie to use them.

To install the tables and create the Cowrie user account enter the following commands:

```
$ mysql -u root -p
CREATE DATABASE cowrie;
GRANT ALL ON cowrie.* TO 'cowrie'@'localhost' IDENTIFIED BY 'PASSWORD HERE';
FLUSH PRIVILEGES;
exit
```

Next create the database schema:

```
$ cd /opt/cowrie/
$ mysql -u cowrie -p
USE cowrie;
source ./docs/sql/mysql.sql;
exit
```

disable MySQL strict mode:

```
$ vi /etc/mysql/conf.d/disable_strict_mode.cnf

[mysqld]
sql_mode=IGNORE_SPACE,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_
↪CREATE_USER,NO_ENGINE_SUBSTITUTION
```

17.4 Cowrie Configuration for Kippo-Graph

Edit cowrie.cfg:

```
$ vi etc/cowrie.cfg
```

Activate output to mysql:

```
[output_mysql]
enabled = true
host = localhost
database = cowrie
username = cowrie
password = PASSWORD HERE
port = 3306
debug = false
```

Set read access to tty-files for group www-data (group maybe differ on other distributions):

```
$ sudo apt-get install acl
$ sudo setfacl -Rm g:www-data:rx /opt/cowrie/var/lib/cowrie/tty/
```

17.5 Kippo-Graph Configuration

Edit config file:

```
$ vi /var/www/html/kippo-graph/config.php
```

Change db settings:

```
define('DB_HOST', 'localhost');
define('DB_USER', 'cowrie');
define('DB_PASS', 'PASSWORD HERE');
define('DB_NAME', 'cowrie');
define('DB_PORT', '3306');
```

17.6 Apache2 configuration (optional)

To secure the installation

Create password database:

```
$ cd /etc/apache2/
$ htpasswd -c /etc/apache2/cowrie.passwd <username>
$ htpasswd /etc/apache2/cowrie.passwd <username> (second user)


$ vi /etc/apache2/sites-enabled/000-default.conf
```

Between the <VirtualHost> </VirtualHost> tags, add:

```
<Location />
    AuthBasicAuthoritative On
```

(continues on next page)

(continued from previous page)

```
    AllowOverride AuthConfig

    AuthType Basic
    AuthName "cowrie honeypot"
    AuthUserFile /etc/apache2/cowrie.passwd
    Require valid-user
</Location>
```

CHAPTER

EIGHTEEN

HOW TO SEND COWRIE OUTPUT TO A PROMETHEUS

This guide will show you how to stand up a complete monitoring stack in Docker:

1. Prometheus
2. Cowrie
3. node-exporter (host-level metrics)
4. cAdvisor (container-level metrics)

All containers will join a user-defined Docker network so they can find one another by name.

CHAPTER

NINETEEN

1. CREATE THE DOCKER NETWORK

```
docker network create cowrie-net
```

CHAPTER

TWENTY

2. RUN PROMETHEUS

Create a volume for Prometheus's TSDB

```
docker volume create prometheus-data
```

For configuration file you can

Copy the example config into */etc/prometheus* on your host

```
sudo mkdir -p /etc/prometheus
sudo cp ./docs/prometheus/prometheus.yaml /etc/prometheus/prometheus.yaml
```

Or from ~/cowrie call docker run with updated path

```
-v ./docs/prometheus/prometheus.yaml:/etc/prometheus/prometheus.yaml:ro \
```

CHAPTER

TWENTYONE

3. LAUNCH PROMETHEUS ON *COWRIE-NET*

```
docker run -d \
  --name prometheus \
  --network cowrie-net \
  -p 9090:9090 \
  -v /etc/prometheus/prometheus.yaml:/etc/prometheus/prometheus.yaml:ro \
  -v prometheus-data:/prometheus \
  prom/prometheus \
  --config.file=/etc/prometheus/prometheus.yaml
```

Verify it's running at http://localhost:9090/targets

CHAPTER

TWENTYTWO

3. RUN COWRIE WITH PROMETHEUS METRICS

```
docker run
  --name cowrie \
  --network cowrie-net \
  -p 2222:2222 \
  -p 9000:9000 \
  -e COWRIE_OUTPUT_PROMETHEUS_ENABLED=yes \
  cowrie/cowrie:latest
```

—

CHAPTER

TWENTYTHREE

4. RUN NODE-EXPORTER (HOST METRICS)

```
docker run -d \
  --name node-exporter \
  --network cowrie-net \
  --pid host \
  -v /:/host:ro \
  -p 9100:9100 \
  quay.io/prometheus/node-exporter:latest \
  --path.rootfs /host
```

—

CHAPTER

TWENTYFOUR

5. RUN CADVISOR (CONTAINER METRICS)

```
docker run -d \
  --name cadvisor \
  --network cowrie-net \
  --privileged \
  -v /:/rootfs:ro \
  -v /var/run:/var/run:rw \
  -v /sys:/sys:ro \
  -v /sys/fs/cgroup:/sys/fs/cgroup:ro \
  -v /var/lib/docker/:/var/lib/docker:ro \
  -p 8080:8080 \
  gcr.io/cadvisor/cadvisor:latest
```

CHAPTER

TWENTYFIVE

RUN COWRIE WITH PROMETHEUS LOCALLY

Add the following entries in `etc/cowrie.cfg` under the Output Plugins section:

```
[output_prometheus]
enabled = true
port = 9000
debug = false
```

Ensure your *prometheus.yaml* has:

```
global:
  scrape_interval: 5s
scrape_configs:
  - job_name: 'cowrie'
    static_configs:
      - targets: [
        'localhost:9000',
      ]
  - job_name: 'scrapers'
    static_configs:
      - targets: [
        'node-exporter:9100',
        'cadvisor:8080'
      ]
    metric_relabel_configs:
      - source_labels: [ cowrie ]
        regex: '^cowrie$'
        action: keep
      - action: drop
        regex: '.*'
```

Reload Prometheus if needed, then visit **Status** → **Targets** to confirm all three are UP.

CHAPTER

TWENTYSIX

HOW TO SEND COWRIE OUTPUT TO AZURE SENTINEL

Open your Sentinel worksapce and navigate to *Data connectors* > *Syslog* > *Open connector page*. Expand *Install agent on a non-Azure Linux Machine*, the select *Download & install agent for non-Azure Linux machines*. Select the Linux tab and either copy the shell script that is presented, or take note of your Workspace ID and Primary Key and install the agent on your host by hand:

```
$ wget https://raw.githubusercontent.com/Microsoft/OMS-Agent-for-Linux/master/installer/
↪scripts/onboard_agent.sh
$ chmod +x onboard_agent.sh
$ ./onboard_agent.sh -w <workspace ID> -s <key> -d opinsights.azure.com
```

Once installed, return to the Syslog connector page and select *Open your workspace advanced settings configuration*. Select *Data* > *Custom Logs*. Check *Apply below configuration to my linux machines* then add a new custom log source: When prompted, upload the `cowrie.json` file you downloaded.

The default delimeter is correct (newline). Specify `/opt/cowrie/var/log/cowrie/cowrie.json` as the log collection path.

Name the custom log `cowrie_JSON`. Sentinel will automatically append _CL to this name.

It will take a while for this to roll out to the host, but eventually you'll be able to run the log analytics query cowrie_JSON_CL and see data coming in.

Take the contents of `cowrie-parser.txt` from the `docs/sentinel` folder and paste them into a new log analytics query. Run the query, then save this off as a function with the name, alias and category of *Cowrie*.

Once events are being ingested and parsed by Azure Sentinel, `linux_workbook.json` can be imported to define a custom workbook to interact with Cowrie data.

CHAPTER

TWENTYSEVEN

HOW TO SEND COWRIE OUTPUT TO SPLUNK

27.1 Splunk Output Module

- In Splunk, enable the HTTP Event Collector (go to Settings->Add Data)
- Do not enable *Indexer Acknowledgment*
- Copy the authorization token for later use
- Modify `cowrie.cfg` to enable the `[output_splunk]` section
- Configure the URL for HTTP Event Collector and add the authorization token you copied in the previous step
- Optionally enable sourcetype, source, host and index settings

27.2 File Based

- Collect `var/log/cowrie/cowrie.json` output file using Splunk

27.3 Reporting

Please see: https://github.com/aplura/Tango

CHAPTER

TWENTYEIGHT

HOW TO SEND COWRIE OUTPUT TO A MYSQL OR POSTGRESQL DATABASE

28.1 MySQL/PostgreSQL Output Plugin Prerequisites

- Working Cowrie installation
- Working MySQL installation
- Working PostgreSQL installation

28.2 MySQL Installation

On your Cowrie server, run:

```
$ su - cowrie
$ source cowrie/cowrie-env/bin/activate
$ pip install mysql-connector-python
```

28.3 MySQL Configuration

First create an empty database named `cowrie`:

```
$ mysql -u root -p
CREATE DATABASE cowrie;
```

Create a Cowrie user account for the database and grant all access privileges:

```
CREATE USER 'cowrie'@'localhost' IDENTIFIED BY 'PASSWORD HERE';
```

Restricted Privileges:

Alternatively you can grant the Cowrie account fewer privileges. The following command grants the account with the bare minimum required for the output logging to function:

```
GRANT INSERT, SELECT, UPDATE ON cowrie.* TO 'cowrie'@'localhost';
```

Apply the privilege settings and exit mysql:

```
FLUSH PRIVILEGES;
exit
```

Next, log into the MySQL database using the Cowrie account to verify proper access privileges and load the database schema provided in the docs/sql/ directory:

```
$ cd ~/cowrie/docs/sql/
$ mysql -u cowrie -p
USE cowrie;
source mysql.sql;
exit
```

28.4 Cowrie Configuration for MySQL

Add the following entries to `etc/cowrie.cfg` under the Output Plugins section:

```
[output_mysql]
host = localhost
database = cowrie
username = cowrie
password = PASSWORD HERE
port = 3306
debug = false
enabled = true
```

Restart Cowrie:

```
$ cd ~/cowrie/bin/
$ ./cowrie restart
```

Verify that the MySQL Output Engine Has Been Loaded

Check the end of the `var/log/cowrie/cowrie.log` to make sure the MySQL output engine loaded successfully:

```
$ cd ~/cowrie/var/log/cowrie/
$ tail cowrie.log
```

Example expected output:

```
2017-11-27T22:19:44-0600 [-] Loaded output engine: jsonlog
2017-11-27T22:19:44-0600 [-] Loaded output engine: mysql
...
2017-11-27T22:19:58-0600 [-] Ready to accept SSH connections
```

Confirm that events are logged to the MySQL Database

Wait for a new login attempt to occur. Use tail like before to quickly check if any activity has been recorded in the cowrie.log file.

Once a login event has occurred, log back into the MySQL database and verify that the event was recorded:

```
$ mysql -u cowrie -p
USE cowrie;
SELECT * FROM auth;
``
```

Example output:

```
+----+--------------+---------+----------+-------------+---------------------+
| id | session      | success | username | password    | timestamp           |
+----+--------------+---------+----------+-------------+---------------------+
|  1 | a551c0a74e06 |       0 | root     | 12345       | 2017-11-27 23:15:56 |
|  2 | a551c0a74e06 |       0 | root     | seiko2005   | 2017-11-27 23:15:58 |
|  3 | a551c0a74e06 |       0 | root     | anko        | 2017-11-27 23:15:59 |
|  4 | a551c0a74e06 |       0 | root     | 123456      | 2017-11-27 23:16:00 |
|  5 | a551c0a74e06 |       0 | root     | dreambox    | 2017-11-27 23:16:01 |
...
```

28.5 PostgreSQL Installation

On your Cowrie server, run:

```
$ su - cowrie
$ source cowrie/cowrie-env/bin/activate
$ pip install psycopg2
```

28.6 PostgreSQL Configuration

First create an empty database named `cowrie` as a PostgreSQL superuser (e.g., `postgres`):

```
$ psql -U postgres
CREATE DATABASE cowrie;
```

Create a Cowrie user account for the database and grant access privileges:

```
CREATE USER cowrie WITH PASSWORD 'PASSWORD HERE';
GRANT CONNECT ON DATABASE cowrie TO cowrie;
\c cowrie
GRANT USAGE ON SCHEMA public TO cowrie;
GRANT INSERT, SELECT, UPDATE ON ALL TABLES IN SCHEMA public TO cowrie;
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT INSERT, SELECT, UPDATE ON TABLES TO␣
↪cowrie;
\q
```

Log into the PostgreSQL database using the Cowrie account to verify proper access privileges and load the database schema provided in the `docs/sql/` directory:

```
$ cd ~/cowrie/docs/sql/
$ psql -U cowrie -d cowrie -f postgres.sql
```

28.7 PostgreSQL Schema Update for Boolean Compatibility

PostgreSQL does not support TINYINT. If you are porting the MySQL schema, update boolean-like fields to use PostgreSQL's `BOOLEAN` type or `INTEGER` with 0/1 semantics.

28.8 Cowrie Configuration for PostgreSQL

Add the following entries in `etc/cowrie.cfg` under the Output Plugins section:

```
[output_postgresql]
enabled = true
host = localhost
database = cowrie
username = cowrie
password = PASSWORD HERE
port = 5432
debug = false
```

Restart Cowrie:

```
$ cd ~/cowrie/bin/
$ ./cowrie restart
```

28.9 Verify That the PostgreSQL Output Engine Has Been Loaded

Check the end of the `var/log/cowrie/cowrie.log` to make sure that the PostgreSQL output engine has loaded successfully:

```
$ cd ~/cowrie/var/log/cowrie/
$ tail cowrie.log
```

Example expected output:

```
2025-04-07T22:20:00-0000 [-] Loaded output engine: jsonlog
2025-04-07T22:20:00-0000 [-] Loaded output engine: postgresql
...
2025-04-07T22:20:14-0000 [-] Ready to accept SSH connections
```

28.10 Confirm That Events are Logged to the PostgreSQL Database

Wait for a new login attempt to occur. Use `tail` like before to quickly check if any activity has been recorded in the `cowrie.log` file.

Once a login event has occurred, log back into the PostgreSQL database and verify that the event was recorded:

```
$ psql -U cowrie -d cowrie
SELECT * FROM auth;
```

Example output:

```
 id |     session      | success | username | password  |     timestamp
----+------------------+---------+----------+-----------+---------------------
  1 | 863c26257d88     | t       | root     | 12345     | 2025-04-07 22:23:14
  2 | 863c26257d88     | f       | root     | dreambox  | 2025-04-07 22:23:15
...
```

CHAPTER

TWENTYNINE

USING TCP TUNNELING WITH SQUID

29.1 Squid Prerequisites

- Working Cowrie installation
- Working Squid installation with CONNECT allowed
- Rate limit and black/white lists in Squid (optional)

29.2 Squid Installation

Install Squid:

```
$ sudo apt-get install squid
```

29.3 Squid Configuration

See `squid.conf` for an example configuration.

29.4 Cowrie Configuration for Squid

Uncomment and update the following entries to `etc/cowrie.cfg` under the SSH section:

```
[ssh]
forward_tunnel = true
forward_tunnel_80 = 127.0.0.1:3128
forward_tunnel_443 = 127.0.0.1:3128
```

Restart Cowrie

Restart:

```
$ bin/cowrie restart
```

CHAPTER

THIRTY

AUTOMATICALLY STARTING COWRIE WITH SUPERVISORD

- Copy the file `cowrie.conf` to `/etc/supervisor/conf/`

CHAPTER

THIRTYONE

AUTOMATICALLY STARTING COWRIE WITH SYSTEMD

NOTE: untested

- Copy the file `docs/systemd/system/cowrie.socket` to `/etc/systemd/system`
- Copy the file `docs/systemd/system/cowrie.service` to `/etc/systemd/system`
- Examine `/etc/systemd/system/cowrie.service` and ensure the paths are correct for your installation if you use non-standard file system locations.
- Add entries to `etc/cowrie.cfg` to listen on ports via systemd. These must match your cowrie.socket configuration:

```
[ssh]
listen_endpoints = systemd:domain=INET6:index=0

[telnet]
listen_endpoints = systemd:domain=INET6:index=1
```

- Run:

```
$ sudo systemctl daemon-reload
$ sudo systemctl start cowrie.service
$ sudo systemctl enable cowrie.service
```

CHAPTER

THIRTYTWO

VIRUSTOTAL INTEGRATION

32.1 Overview

The VirusTotal output plugin integrates Cowrie honeypot with VirusTotal's v3 API to automatically scan and track malicious files and URLs captured by the honeypot.

32.2 Features

- **File Scanning**: Automatically check downloaded files against VirusTotal's database
- **File Uploading**: Upload new malware samples to VirusTotal for analysis
- **URL Scanning**: Check malicious URLs for existing reports
- **URL Submission**: Submit new URLs for scanning
- **Comments**: Add Cowrie attribution comments to files and URLs (with #Cowrie hashtag)
- **Collections**: Organize all Cowrie artifacts in a dedicated VirusTotal collection for easy tracking

32.3 Prerequisites

1. **VirusTotal API Key**: Sign up for a free API key at https://www.virustotal.com/
2. **API Rate Limits**: Be aware of API rate limits (free tier: 4 requests/minute, 500 requests/day)

32.4 Configuration

Add the following to your `etc/cowrie.cfg` file:

```
[output_virustotal]
enabled = true

# Your VirusTotal API key (required)
api_key = 0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef

# Upload new files to VirusTotal (default: True)
upload = true

# Enable debug logging (default: False)
debug = false
```

(continues on next page)

(continued from previous page)

```
# Scan downloaded files (default: True)
scan_file = true

# Scan URLs (default: True)
# Note: This doubles API requests for downloads
scan_url = true

# Optional: Collection name for organizing artifacts
# If not set, no collection will be created
collection = cowrie

# Optional: Custom comment text (default: Cowrie attribution)
#commenttext = First seen by #Cowrie SSH/telnet Honeypot http://github.com/cowrie/cowrie
```

32.5 Configuration Options

32.5.1 api_key (required)

Your VirusTotal API key. Get one at https://www.virustotal.com/gui/my-apikey

32.5.2 upload (optional, default: True)

Whether to upload new files to VirusTotal. If `false`, only existing reports will be retrieved.

32.5.3 scan_file (optional, default: True)

Enable scanning of downloaded files. When enabled:

1. File hash is checked against VirusTotal database
2. If not found and `upload=true`, file is uploaded for analysis
3. If upload is successful and `comment=true`, a comment is added
4. If collection is configured, file is added to the collection

32.5.4 scan_url (optional, default: True)

Enable scanning of URLs from which files were downloaded. **Note**: This doubles the number of API requests for each file download event.

When enabled:

1. URL is checked against VirusTotal database
2. If not found, URL is submitted for scanning
3. If submission is successful and `comment=true`, a comment is added
4. If collection is configured, URL is added to the collection

32.5.5 comment (optional, default: True)

Add a comment to newly uploaded files and submitted URLs. The comment includes:

- Cowrie attribution
- Link to Cowrie GitHub repository

- `#Cowrie` hashtag for easy searching in VirusTotal

32.5.6 collection (optional, default: None)

Name of a VirusTotal collection to organize all Cowrie artifacts. When set:

- Collection is automatically created on Cowrie startup (if it doesn't exist)
- All uploaded files are added to the collection
- All submitted URLs are added to the collection
- Provides centralized view of all honeypot findings in VirusTotal

Benefits:

- Track all artifacts in one place
- Share collection with other researchers
- Monitor honeypot activity over time
- Export IOCs from the collection

If not set or commented out, no collection operations will be performed.

32.5.7 debug (optional, default: False)

Enable verbose logging for troubleshooting. Shows:

- Full API request/response details
- Collection operations
- Error details

32.6 Output Events

The plugin logs events with the following event IDs:

32.6.1 cowrie.virustotal.scanfile

Logged when a file scan result is retrieved.

Attributes:

- `session`: Cowrie session ID
- `sha256`: File SHA-256 hash
- `positives`: Number of antivirus engines detecting file as malicious
- `total`: Total number of antivirus engines
- `scan_date`: Date when file was last scanned
- `scans`: Dictionary of per-engine scan results
- `is_new`: `"true"` if file was newly uploaded, `"false"` if existing

32.6.2 cowrie.virustotal.scanurl

Logged when a URL scan result is retrieved.

Attributes:

- `session`: Cowrie session ID
- `url`: The scanned URL
- `positives`: Number of engines flagging URL as malicious
- `total`: Total number of engines
- `scan_date`: Date when URL was last scanned
- `scans`: Dictionary of per-engine scan results
- `is_new`: `"true"` if URL was newly submitted, `"false"` if existing

32.7 Example Output

File scan result (existing file):

```
{
  "eventid": "cowrie.virustotal.scanfile",
  "session": "abc123def456",
  "sha256": "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
  "positives": 45,
  "total": 70,
  "scan_date": "2025-10-08T12:34:56Z",
  "is_new": "false",
  "scans": {
    "avast": {"detected": "true", "result": "trojan.generic"},
    "kaspersky": {"detected": "true", "result": "trojan.win32.generic"}
  }
}
```

URL scan result (newly submitted):

```
{
  "eventid": "cowrie.virustotal.scanurl",
  "session": "abc123def456",
  "url": "http://malicious-site.example.com/payload.exe",
  "positives": 12,
  "total": 68,
  "scan_date": "2025-10-08T12:35:00Z",
  "is_new": "true"
}
```

32.8 Collections

Collections organize related files and URLs in VirusTotal for better tracking and analysis.

32.8.1 Setting Up a Collection

1. Add `collection = cowrie` to your configuration
2. Restart Cowrie
3. Collection will be automatically created on first startup
4. All subsequent uploads/submissions will be added to this collection

32.8.2 Accessing Your Collection

View your collection in VirusTotal:

`https://www.virustotal.com/gui/collection/<collection-id>`

The collection ID is logged when the collection is created.

32.8.3 Collection Features

- **Centralized View**: All honeypot artifacts in one place
- **Search & Filter**: Filter by file type, detection rate, submission date
- **Export**: Export IOCs in various formats (STIX, CSV, etc.)
- **Sharing**: Share collection with other researchers (enterprise feature)
- **Comments**: View all Cowrie-attributed comments on items

32.9 Best Practices

32.9.1 Rate Limiting

Free VirusTotal API tier has strict rate limits:

- 4 requests per minute
- 500 requests per day

Recommendations:

- Set `scan_url = false` to reduce API usage (disabling URL scans halves the API requests)
- Monitor your API usage in VirusTotal dashboard
- Consider upgrading to paid tier for high-traffic honeypots

32.9.2 Duplicate Prevention

Cowrie automatically prevents duplicate scans using:

- **File deduplication**: Files are only scanned once per download (based on modification time)
- **URL caching**: URLs are cached to prevent repeated lookups within the same session

32.9.3 Collection Management

- Use descriptive collection names (e.g., `cowrie-prod`, `cowrie-lab`)
- Create separate collections for different honeypot deployments
- Regularly review collection contents in VirusTotal

32.9.4 Privacy Considerations

Important: Files and URLs uploaded to VirusTotal become **publicly accessible** to all VirusTotal users.

- Do not upload sensitive files
- Be aware that malware samples will be shared with security community
- URLs will be visible to other researchers

32.10 Troubleshooting

32.10.1 API Key Errors

```
VT scanfile failed: 401 Unauthorized
```

Solution: Verify your API key is correct in `cowrie.cfg`

32.10.2 Rate Limit Errors

```
VT: Error - QuotaExceededError: Quota exceeded
```

Solutions:

- Wait for rate limit to reset (1 minute for request limit, 24 hours for daily limit)
- Reduce scan frequency by disabling `scan_url`
- Upgrade to paid API tier

32.10.3 Collection Already Exists

```
VT: Collection 'cowrie' already exists - will use existing
```

Not an error: This is normal behavior. The plugin will reuse the existing collection.

32.10.4 Debug Mode

Enable debug mode to see detailed API interactions:

```
[output_virustotal]
debug = true
```

This will log full request/response details to help diagnose issues.

32.11 API Reference

The plugin uses VirusTotal v3 API endpoints:

- `GET /api/v3/files/{hash}` - Retrieve file scan report
- `POST /api/v3/files` - Upload new file
- `POST /api/v3/files/{id}/comments` - Add comment to file
- `GET /api/v3/urls/{url_id}` - Retrieve URL scan report
- `POST /api/v3/urls` - Submit URL for scanning

- `POST /api/v3/urls/{id}/comments` - Add comment to URL
- `POST /api/v3/collections` - Create collection
- `POST /api/v3/collections/{id}/files` - Add file to collection
- `POST /api/v3/collections/{id}/urls` - Add URL to collection

For complete API documentation, see: https://docs.virustotal.com/reference/overview

32.12 Contributing

To contribute improvements to the VirusTotal integration:

1. Fork the Cowrie repository
2. Create a feature branch
3. Make your changes to `src/cowrie/output/virustotal.py`
4. Add tests to `src/cowrie/test/test_virustotal.py`
5. Run the test suite: `python -m pytest src/cowrie/test/test_virustotal.py -v`
6. Run type checking: `mypy src/cowrie/output/virustotal.py`
7. Run linting: `ruff check src/cowrie/output/virustotal.py`
8. Submit a pull request

32.13 License

The VirusTotal integration is part of Cowrie and is licensed under the same BSD license.

32.14 Support

- **Documentation**: https://cowrie.readthedocs.io/
- **Issues**: https://github.com/cowrie/cowrie/issues
- **Discussions**: https://github.com/cowrie/cowrie/discussions
- **VirusTotal Support**: https://support.virustotal.com/

CHAPTER

THIRTYTHREE

INDICES AND TABLES

- genindex
- modindex
- search