cowrie Documentation

Release 2.5.0

Michel Oosterhof

Jan 09, 2023

CONTENTS:

COWRIE

1.1 Welcome to the Cowrie GitHub repository

This is the official repository for the Cowrie SSH and Telnet Honeypot effort.

1.2 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.

Cowrie is maintained by Michel Oosterhof.

1.3 Documentation

The Documentation can be found here.

1.4 Slack

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

1.5 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

For both settings:

- Session logs are stored in an UML Compatible format for easy replay with the *bin/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

1.6 Docker

Docker versions are available.

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

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

- On Docker Hub: https://hub.docker.com/r/cowrie/cowrie

- Configuring Cowrie in Docker

Cowrie in Docker can be configured using environment variables. The variables start with COWRIE then have the section name in capitals, followed by the stanza in capitals. An example is below to enable telnet support:

```
COWRIE_TELNET_ENABLED=yes
```

Alternatively, Cowrie in Docker can use an *etc* volume to store configuration data. Create *cowrie.cfg* inside the etc volume with the following contents to enable telnet in your Cowrie Honeypot in Docker:

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

1.7 Requirements

Software required to run locally:

- Python 3.8+

- python-virtualenv

For Python dependencies, see requirements.txt.

1.8 Files of interest:

- *etc/cowrie.cfg* - Cowrie's configuration file. Default values can be found in etc/cowrie.cfg.dist.
- *share/cowrie/fs.pickle* - fake filesystem
- *etc/userdb.txt* - credentials to access the honeypot
- honeyfs/ - file contents for the fake filesystem - feel free to copy a real system here or use *bin/fsctl*
- *honeyfs/etc/issue.net* - pre-login banner
- honeyfs/etc/motd - post-login banner
- *var/log/cowrie/cowrie.json* - transaction output in JSON format
- *var/log/cowrie/cowrie.log* - log/debug output
- *var/lib/cowrie/tty/* - session logs, replayable with the *bin/playlog* utility.
- *var/lib/cowrie/downloads/* - files transferred from the attacker to the honeypot are stored here
- share/cowrie/txtcmds/ - file contents for simple fake commands
- bin/createfs - used to create the fake filesystem
- bin/playlog - utility to replay session logs

1.9 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.

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 *master* 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.

LICENSE

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 users is called *phil* these days. Having the same user always available is an easy way to identify Cowrie so it's recommend to change this setup. You can modify it by doing the following:

```
$ vi honeyfs/etc/passwd
```

And edit the userid. Then:

```
$ bin/fsctl share/cowrie/fs.pickle
fs.pickle:/$ mv /home/phil /home/joe
```

And then restart Cowrie:

```
$ bin/cowrie restart
```

4.4 How do I add files to the file system?

The file system meta data is stored in the pickle file. The file contents is stored in the *honeyfs* directory. To add a file, the minimum action is to modify the pickle file. Doing this makes the file show up in *ls* and other commands. But it won't have any contents available. To add file contents, you'll need a file to honeyfs.

First add a file system entry, the *1024* here is the file size. The *chown* commands only takes numerical uid's, they should match entries in *honeyfs/etc/passwd*:

```
$ bin/fsctl share/cowrie/fs.pickle
fs.pickle:/$ touch /home/phil/myfile 1024
fs.pickle:/$ chown 1000:1000 /home/phil/myfile
```

Then create or copy a file in the *honeyfs*:

```
$ cp myfile /honeyfs/home/phil
```

INSTALLING COWRIE IN SEVEN STEPS

This guide describes how to install Cowrie in `shell` mode. For `proxy` mode read *PROXY.rst*.

- *Step 1: Install system dependencies*
- *Step 2: Create a user account*
- *Step 3: Checkout the code*
- *Step 4: Setup Virtual Environment*
- *Step 5: Install configuration file*
- *Step 6: Starting Cowrie*
- *Step 7: Listening on port 22 (OPTIONAL)*
- *Installing Backend Pool dependencies (OPTIONAL)*
- *Running using supervisord (OPTIONAL)*
- *Configure Additional Output Plugins (OPTIONAL)*
- *Troubleshooting*
- *Updating Cowrie*
- *Modifying Cowrie*

5.1 Step 1: Install system dependencies

First we install system-wide support for Python virtual environments and other dependencies. Actual Python packages are installed later.

On Debian based systems (last verified on Debian 10, 2021-04-29):

```
$ sudo apt-get install git python3-virtualenv libssl-dev libffi-dev build-essential␣
→libpython3-dev python3-minimal authbind virtualenv
```

5.2 Step 2: Create a user account

It's strongly recommended to run with a dedicated non-root user id:

```
$ sudo adduser --disabled-password cowrie
Adding user 'cowrie' ...
Adding new group 'cowrie' (1002) ...
Adding new user 'cowrie' (1002) with group 'cowrie' ...
Changing the user information for cowrie
Enter the new value, or press ENTER for the default
Full Name []:
Room Number []:
Work Phone []:
Home Phone []:
Other []:
Is the information correct? [Y/n]

$ sudo su - cowrie
```

5.3 Step 3: Checkout the code

Check out the code:

```
$ git clone http://github.com/cowrie/cowrie
Cloning into 'cowrie'...
remote: Counting objects: 2965, done.
remote: Compressing objects: 100% (1025/1025), done.
remote: Total 2965 (delta 1908), reused 2962 (delta 1905), pack-reused 0
Receiving objects: 100% (2965/2965), 3.41 MiB | 2.57 MiB/s, done.
Resolving deltas: 100% (1908/1908), done.
Checking connectivity... done.

$ cd cowrie
```

5.4 Step 4: Setup Virtual Environment

Next you need to create your virtual environment:

```
$ pwd
/home/cowrie/cowrie
$ python -m venv cowrie-env
New python executable in ./cowrie/cowrie-env/bin/python
Installing setuptools, pip, wheel...done.
```

Activate the virtual environment and install packages:

```
$ source cowrie-env/bin/activate
(cowrie-env) $ python -m pip install --upgrade pip
(cowrie-env) $ python -m pip install --upgrade -r requirements.txt
```

5.5 Step 5: Install configuration file

The configuration for Cowrie is stored in cowrie.cfg.dist and cowrie.cfg (Located in cowrie/etc). Both files are read on startup, where entries from cowrie.cfg take precedence. The .dist file can be overwritten by upgrades, cowrie.cfg will not be touched. To run with a standard configuration, there is no need to change anything. To enable telnet, for example, create cowrie.cfg and input only the following:

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

5.6 Step 6: Starting Cowrie

Start Cowrie with the cowrie command. You can add the cowrie/bin directory to your path if desired. An existing virtual environment is preserved if activated, otherwise Cowrie will attempt to load the environment called "cowrie-env":

```
$ bin/cowrie start
Activating virtualenv "cowrie-env"
Starting cowrie with extra arguments [] ...
```

5.7 Step 7: Listening on port 22 (OPTIONAL)

There are three methods to make Cowrie accessible on the default SSH port (22): *iptables*, *authbind* and *setcap*.

5.7.1 Iptables

Port redirection commands are system-wide and need to be executed as root. A firewall redirect can make your existing SSH server unreachable, remember to move the existing server to a different port number first.

The following firewall rule will forward incoming traffic on port 22 to port 2222 on Linux:

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

Or for telnet:

```
$ sudo iptables -t nat -A PREROUTING -p tcp --dport 23 -j REDIRECT --to-port 2223
```

Note that you should test this rule only from another host; it doesn't apply to loopback connections.

On MacOS run:

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

5.7.2 Authbind

Alternatively you can run authbind to listen as non-root on port 22 directly:

```
$ 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
```

Edit bin/cowrie and modify the AUTHBIND_ENABLED setting

Change the listening port to 22 in cowrie.cfg:

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

Or for telnet:

```
$ apt-get install authbind
$ sudo touch /etc/authbind/byport/23
$ sudo chown cowrie:cowrie /etc/authbind/byport/23
$ sudo chmod 770 /etc/authbind/byport/23
```

Change the listening port to 23 in cowrie.cfg:

```
[telnet]
listen_endpoints = tcp:2223:interface=0.0.0.0
```

5.7.3 Setcap

Or use setcap to give permissions to Python to listen on ports<1024:

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

And change the listening ports in `cowrie.cfg` as above.

5.8 Installing Backend Pool dependencies (OPTIONAL)

If you want to use the proxy functionality combined with the automatic backend pool, you need to install some dependencies, namely QEMU, libvirt, and their Python interface. In Debian/Ubuntu:

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

Then install the Python API to run the backend pool:

```
(cowrie-env) $ python -m pip install libvirt-python==6.4.0
```

To allow QEMU to use disk images and snapshots, set it to run with the user and group of the user running the pool (usually called 'cowrie' too:

```
$ sudo vim /etc/libvirt/qemu.conf
```

Search and set both *user* and *group* to *"cowrie"*, or the username/group you'll be running the backend pool with.

5.9 Running using Supervisord (OPTIONAL)

On Debian, put the below in /etc/supervisor/conf.d/cowrie.conf:

```
[program:cowrie]
command=/home/cowrie/cowrie/bin/cowrie start
directory=/home/cowrie/cowrie/
user=cowrie
autorestart=true
redirect_stderr=true
```

Update the bin/cowrie script, change:

```
DAEMONIZE=""
```

to:

```
DAEMONIZE="-n"
```

5.10 Configure Additional Output Plugins (OPTIONAL)

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

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

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

5.11 Troubleshooting

5.11.1 CryptographyDeprecationWarning: Blowfish has been deprecated

The following warnings may occur, these can be safely ignored, and are not the reason your Cowrie installation is not working:

```
CryptographyDeprecationWarning: Blowfish has been deprecated b"blowfish-cbc":␣
→(algorithms.Blowfish, 16, modes.CBC),
CryptographyDeprecationWarning: CAST5 has been deprecated b"cast128-cbc": (algorithms.
→CAST5, 16, modes.CBC),
CryptographyDeprecationWarning: Blowfish has been deprecated b"blowfish-ctr":␣
→(algorithms.Blowfish, 16, modes.CTR),
CryptographyDeprecationWarning: CAST5 has been deprecated b"cast128-ctr": (algorithms.
→CAST5, 16, modes.CTR),
```

The algorithms are used in Cowrie to support old attackers that use these deprecated algorithms.

5.11.2 twistd: unknown command: cowrie

If you see `twistd:  Unknown command:  cowrie` there are two possibilities. If there's a Python stack trace, it probably means there's a missing or broken dependency. If there's no stack trace, double check that your PYTHONPATH is set to the source code directory.

5.11.3 Default file permissions

To make Cowrie logfiles public readable, change the `--umask 0077` option in `bin/cowrie` into `--umask 0022`

5.11.4 General approach

Check the log file in `var/log/cowrie/cowrie.log`.

5.12 Updating Cowrie

First stop your honeypot. Then pull updates from GitHub, and upgrade your Python dependencies:

```
$ bin/cowrie stop
$ git pull
$ python -m pip install --upgrade -r requirements.txt
```

If you use output plugins like SQL, Splunk, or ELK, remember to also upgrade your dependencies for these too:

```
$ python -m pip install --upgrade -r requirements-output.txt
```

And finally, restart Cowrie after finishing all updates:

```
$ bin/cowrie start
```

5.13 Modifying Cowrie

The pre-login banner can be set by creating the file `honeyfs/etc/issue.net`. The post-login banner can be customized by editing `honeyfs/etc/motd`.

RELEASE NOTES

6.1 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

6.2 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)

6.3 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

6.4 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

6.5 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.

6.6 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

6.7 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

6.8 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.

6.9 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

6.10 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

6.11 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

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.

CHANGING THE COWRIE FILE SYSTEM

8.1 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 deleted when they log off. They can delete or change any file, nothing will be preserved.

The file system implementation consists of two pieces, the *pickle* file, which holds metadata for the files (filename, directory, permissions, owner, size, file type, etc), and the *honeyfs* directory that holds file contents. The honeyfs directory only has certain files by default. Most files do not have content associated with them.

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

8.2 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:

```
$ ./bin/createfs -l YOUR-DIR -d DEPTH -o ./share/cowrie/custom.pickle
```

Make sure your config picks up custom.pickle, or rename it to fs.pickle

The pickle file just gives the layout of filesystem. If you actually need the files to be really there (for example to read them), you'll have to create them in honeyfs/xxx

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.

9.1 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"
```

9.2 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.

9.3 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.

9.4 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.

9.5 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`.

9.5.1 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.

9.5.2 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.

9.5.3 XML configs (advanced)

You can change libvirt's XML configs from the default ones in `share/cowrie/pool_configs`. However, if you're using one of the default images provided, then you probably don't need to.

9.5.4 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`)

9.5.5 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.

9.6 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.

9.6.1 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
```

9.6.2 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!**

9.7 References

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

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.

10.1 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
```

(continues on next page)

```
= - 0644            0 /var/lib/systemd/timers/stamp-fstrim.timer
= - 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
```

10.2 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

$ 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.

ELEVEN

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.

11.1 Reference

11.1.1 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

11.1.2 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

11.1.3 cowrie.login.success

Successful authentication.

Attributes:

- username

- password

11.1.4 cowrie.login.failed

Failed authentication.

Attributes:

- username

- password

11.1.5 cowrie.client.size

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

Attributes:

- width

- height

11.1.6 cowrie.session.file_upload

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

Attributes:

- filename

- outfile

- shasum

11.1.7 cowrie.command.input

Command line input

Attributes:

- input

11.1.8 cowrie.virustotal.scanfile

File sent to VT for scanning

Attributes:

- sha256

- is_new

- positives

- total

11.1.9 cowrie.session.connect

New connection

Attributes:

- src_ip

- src_port

- dst_ip

- dst_port

11.1.10 cowrie.client.version

SSH identification string

Attributes:

- version

11.1.11 cowrie.client.kex

SSH Key Exchange Attributes

Attributes:

- hassh

- hasshAlgorithms

- kexAlgs

- keyAlgs

11.1.12 cowrie.session.closed

Session closed

Attributes:

- duration

11.1.13 cowrie.log.closed

TTY Log closed

Attributes:

- *duration*: duration of session in seconds
- *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

11.1.14 cowrie.direct-tcpip.request

Request for proxying via the honeypot

Attributes:

- dst_ip
- dst_port
- src_ip
- src_port

11.1.15 cowrie.direct-tcpip.data

Data attempted to be sent through direct-tcpip forwarding

Attributes:

- dst_ip
- dst_port

11.1.16 cowrie.client.var

Attributes:

- name
- value

HOW TO SEND COWRIE OUTPUT TO DATADOG LOG MANAGEMENT

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

12.1 Datadog output module Prerequisites

- Working Cowrie installation
- Existing Datadog account.

12.2 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`.

12.3 Datadog Configuration

JSON logs are handled without further configuration in Datadog.

THIRTEEN

DOCKER REPOSITORY

For Docker please see https://github.com/cowrie/docker-cowrie/

HOW TO SEND COWRIE OUTPUT TO AN ELK STACK

14.1 ElasticSearch Prerequisites

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

14.2 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
```

14.3 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.

14.4 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*)

14.5 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
```

14.6 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
```

14.7 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;
```

(continues on next page)

```
        }
}
```

Start the service:

```
$ sudo systemctl start nginx
```

14.8 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.

14.9 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.

14.10 ElasticSearch Troubleshooting

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

HOW TO SEND COWRIE OUTPUT TO GRAYLOG

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

15.1 Prerequisites

- Working Cowrie installation
- Working Graylog installation

15.2 Cowrie Configuration

15.2.1 Using Syslog

Open the Cowrie configuration file and uncomment these 3 lines:

```
[output_localsyslog]
facility * USER
format * text
```

Restart Cowrie

15.2.2 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

15.3 Graylog Configuration

15.3.1 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.**

15.3.2 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.

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

```json
{
  "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"
}
```

Then click **Launch.**

Note:

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

15.4 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
```

HOW TO SEND COWRIE OUTPUT TO KIPPO-GRAPH

16.1 Kippo-Graph Prerequisites

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

16.2 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

16.3 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
```

16.4 Cowrie Configuration for Kippo-Graph

Edit cowrie.cfg:

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

Activate output to mysql:

```
[output_mysql]
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/
```

16.5 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');
```

16.6 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
    AllowOverride AuthConfig

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

SEVENTEEN

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.

EIGHTEEN

HOW TO SEND COWRIE OUTPUT TO SPLUNK

18.1 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

18.2 File Based

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

18.3 Reporting

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

HOW TO SEND COWRIE OUTPUT TO A MYSQL DATABASE

19.1 MySQL Output Plugin Prerequisites

- Working Cowrie installation
- Working MySQL installation

19.2 MySQL Installation

On your Cowrie server, run:

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

19.3 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:

```
GRANT ALL ON cowrie.* TO 'cowrie'@'localhost' IDENTIFIED BY 'PASSWORD HERE';
```

Restricted Privileges:

Alternatively you can grant the Cowrie account with less 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' IDENTIFIED BY 'PASSWORD
→HERE';
```

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
```

19.4 Cowrie Configuration for MySQL

Uncomment and update 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 ~/cowrie/var/log/cowrie/cowrie.log to make sure that the MySQL output engine has 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 |
...
```

USING TCP TUNNELING WITH SQUID

20.1 Squid Prerequisites

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

20.2 Squid Installation

Install Squid:

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

20.3 Squid Configuration

See `squid.conf` for an example configuration.

20.4 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
```

AUTOMATICALLY STARTING COWRIE WITH SUPERVISORD

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

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
```

HOW TO SEND COWRIE OUTPUT TO THREAT JAMMER

23.1 Threat Jammer Prerequisites

- Working Cowrie installation
- A Threat Jammer API Key

23.2 Threat Jammer Introduction

23.2.1 What is Threat Jammer

Threat Jammer is a service to access high-quality threat intelligence data from a variety of sources and integrate it into their applications with the sole purpose of detecting and blocking malicious activity. This output module feeds the private denylists of Threat Jammer with successful and unsuccessful login attempts thanks to the asynchronous Reporting API of Threat Jammer. The user can use the denylists in their own infrastructure firewalls or WAFs, or use it for forensic analysis.

When users feed Threat Jammer with their data using the Reporting API, they increase the chance of finding malicious activity and reducing the false positives. Crowdsourced Intelligence means that our users can report their data to the Threat Jammer system, and Threat Jammer will assess the data and decide whether aggregate the data to the crowdsourced intelligence database. If the data has enough quality and is ready to consume, Threat Jammer will share it with the community.

Anybody can sign up and use the service for free forever.

23.3 Cowrie Configuration for Threat Jammer

23.3.1 Quick start

To use the output module the user must sign up and obtain an API KEY. The sign-up is free forever.

Once the developers have obtained an API Key, they have to open the configuration file of cowrie:

```
[output_threatjammer]
enabled = false
bearer_token = THREATJAMMER_API_TOKEN
#api_url=https://dublin.report.threatjammer.com/v1/ip
#track_login = true
#track_session = false
#ttl = 86400
#category = ABUSE
#tags = COWRIE,LOGIN,SESSION
```

and edit the section `[output_threatjammer]` as follows:

- Change the `enabled` parameter to `true`.

- Change the `THREATJAMMER_API_TOKEN` placeholder with the real API Key of the user in the `bearer_token` parameter.

[Re]start Cowrie. If the output module was initialized succeessfully, the user will see the following message in the Cowrie log:

```
[-] ThreatJammer.com output plugin successfully initialized. Category=ABUSE. TTL=86400.
→Session Tracking=False. Login Tracking=True
```

23.3.2 Other parameters

The following parameters can change how the output module works, so use it with caution:

api_url

Points to the server running the Report API. By default it points to the Dublin Region, but more regions will be deployed in the future.

track_login

Send information about the IP addresses that tried to login successfully or not. By default is `true`.

track_session

Send information about the IP addresses that created a session. By default is `false`.

ttl

Time to Live in seconds of the reported IP addresses in the private denylist in Threat Jammer. When the TTL expires, the IP address will be automatically removed. If the same IP address is reported more than once, the TTL resets. The default value is `86400`.

category

The category to classify the IP address. By default is `ABUSE`. See Platform Datasets for the list of categories.

tags

Comma separated list of tags to classify the IP addresses. Must be alphanumeric and uppercase. By default, `COWRIE,LOGIN,SESSION`.

23.3.3 Rate limits and buffers

The output module will send the information every 60 seconds or if the buffer of pending IP addresses to send to the Report API is 1000. The first condition will trigger the send action to the Report API server.

There is a limit of 6 hits per minute per API Key. If the limit is reached, the service returns a 429 response code. A single honeypot should never trigger the rate limit.

23.4 About the Module

23.4.1 Python and OS versions

The code has passed the tests implemented in the CI workflows, as expected. This module is compatible with versions of Python from v3.7 and up.

The code does not use any library not already present in the project. It uses `twisted` extensively to communicate with the server.

The code has been extensively tested with the Docker build files with buster-slim and bullseye-slim provided by the project. It was also tested on an Ubuntu 20.04 with Python 3.8.

Minimal testing was done using other versions of Python, and no other operating systems were used throughout the tests. This plugin is thus a beta version.

TWENTYFOUR

INDICES AND TABLES

- genindex
- modindex
- search