Advanced Cowrie Configuration to Increase Honeypot Deceptiveness

Warren Z. Cabral, Craig Valli, Leslie F. Sikos, Samuel G. Wakeling

Advanced Cowrie Configuration to Increase Honeypot Deceptiveness

Warren Z Cabral[1,2[0000-0002-5270-0020]], Craig Valli[1,2[0000-0002-2298-9791]], Leslie F Sikos[1,2[0000-0003-3368-2215]], Samuel G Wakeling[2[0000-0002-7675-5659]]

[1] Cyber Security Co-operative Research Centre, Perth, WA, Australia
[2] Security Research Institute, Edith Cowan University, Perth, WA, Australia

Abstract. Cowrie is a medium-interaction SSH, and Telnet honeypot used to record brute force attacks and SSH requests. Cowrie utilizes a Python codebase, which is maintained and publicly available on GitHub. Since its source code is publicly released, not only security specialists but cybercriminals can also analyze it. Nonetheless, cybersecurity specialists deploy most honeypots with default configurations. This outcome is because modern computer systems and infrastructures do not provide a standard framework for optimal deployment of these honeypots based on the various configuration options available to produce a non-default configuration. This option would allow them to act as effective deceptive systems. Honeypot deployments with default configuration settings are easier to detect because cybercriminals have known scripts and tools such as NMAP and Shodan for identifying them. This research aims to develop a framework that enables for the customized configuration of the Cowrie honeypot, thereby enhancing its functionality to achieve a high degree of deceptiveness and realism when presented to the Internet. A comparison between the default and configured deployments is further conducted to prove the modified deployments' effectiveness.

Keywords: Cybersecurity, Honeypots, Deception, Cowrie, SSH.

1 Introduction

Honeypots are viable tools for network monitoring; to detect, record and analyze attacks in a network. Cybersecurity specialists and academic researchers use a collection of honeypots, i.e., honeynets – to monitor cyberattacks across networks and the entire cyberspace [1, 2]. They reveal an attackers IP, location, commands entered and can also be used to study targeted vulnerabilities and capture malware binaries. This research focuses on the Cowrie honeypot. *Cowrie*[1] interacts with cybercriminals within a simulated SSH environment. Simulation is a biology-inspired deceptive process through which the honeypot provides a service that functions like the actually expected system. For example, being an SSH honeypot, Cowrie simulates an SSH server's behaviour and usually a Linux operating system (It can also be configured to sim-

[1] https://github.com/cowrie/cowrie

ulate a Windows environment exposing SSH to a PowerShell session), thereby allowing cybercriminals to log into the system. Still, every command is only imitated and logged but not actually executed [3]. Hence, cybercriminals can easily detect the difference between a simulated session (honeypot environment) and a legitimate OS session [4]. High-interaction honeypots are challenging to identify because they provide the entire operating system to the cybercriminal rather than a simulated environment [5], and they typically are expensive to operate. Because of high operational costs, many honeypot operators rely on low-interaction or medium-interaction open-source honeypots to simulate vulnerable network services such as SSH, ICMP, and SCADA services. This is because they are not only cost-effective but are easily deployable and maintainable. However, the usage of this method of simulation also entails a number of operational weaknesses; poor deceptive capability and use of default deployments due to an absence of a standard deployment architecture and lack of knowledge on how to deploy honeypots securely and completely. Cybercriminals rely on numerous tools and strategies to detect honeypots these are:

i. *The mental mindset:* "Honeypot detection lies in a cybercriminal's ability to detect and find out about the deceptive nature of the honeypot". Tsikerdekis [6] wrote this, implying that a honeypot's detection lies in what a cybercriminal thinks about how a honeypot typically functions and interacts. This means a cybercriminal may believe that they are attacking a honeypot when it is the real system, or they think they are attacking the existing network but are instead interacting with a honeypot. Therefore, at times honeypot detection depends on the cybercriminal's thinking. For example, the honeypot environment may not be configured correctly; there may be a lack or absence of security mechanisms securing the honeypot. Therefore, the cybercriminal may realize the target is too easy and hence it is a possible trap. Another example is if a honeypot has a balance in functionality, accessibility, and interaction, the cybercriminal may engage with it thinking it is an actual system.

ii. *Default configurations*: As previously mentioned, honeypot deployments with default configuration settings are easier to detect and must be configured before deployment. According to Vetterl [7], the main cause for convenient and easy honeypot detection is that honeypot operators deploy honeypots with default configurations. Their research aimed specifically at Kippo and Cowrie honeypots. It was discovered that 72% of honeypots were deployed with default values. To confirm their hypothesis, the `uname-a` command was executed on the detected honeypots. The command returned the string `Linux [hostname] 3.2.0-4-amd64 #1 SMP Debian 3.2.68-1+deb7u1 x86 64 GNU/Linux\r\n` [8], which is indeed a default value for the Cowrie honeypot instance. Other features that determine honeypot configurations are default hostnames, banners, lack of functionality, insufficient input commands, and standard outputs.

iii. *NMAP*[2]: is a network mapping tool used for detecting and identifying open ports and services [9]. Default instances of Cowrie reveal port 2222 operating on a default Cowrie SSH string. NMAP can also detect honeypots deployed within virtual environments. For example, static MAC addresses for the Cowrie honeypot deployed within a virtual machine [3].

iv. *Shodan*[3]: a scanning service developed by Matherly [10] used to "crawl" the Internet. When analyzing a device, Shodan queries the device's banners, IP address, protocols, port number, hostname, Internet Service Provider host and Geolocational IP information. It can also scan an entire IPv4 Internet address space and therefore is a solid indicator of what can be detected by third parties conducting reconnaissance scanning. Shodan then relates the scanned information into a freely searchable, longitudinally stored database. This database is indexed and searchable by numerous filters such as the ones mentioned above or a city, country, dates, or a device's operating system [11].

v. *Automated scripts*: some cybercriminals have automated scripts developed that can be executed to detect honeypots based on type, settings, and other default values. For example, the python script `cowrie_detect.py`[4] scans the network by collecting hostnames, ports, default arguments and accurately displays if the monitored machine is operating a Cowrie instance. This detection can be mitigated by modifying the default configuration files of the honeypot.

Over 19,208 open-source honeypots were detected using simple tools and signatures. The honeypots were concentrated across 637 autonomous systems, including research, enterprise, cloud and hosting networks [2]. This outcome is because these systems received no configuration before deployment and because cybercriminals are already aware of their default configurations [3, 7]. This research aims to mitigate this problem by modifying these configurations by studying the Cowrie honeypot's different artefacts and changing them to increase the honeypot's overall deceptiveness. This study further intends to educate security specialists and researchers on the differences between default and configured honeypot instances, i.e., centred on their deceptive ability and interaction with cybercriminals, it is critical to configure honeypots before deployment.

[2] https://nmap.org
[3] https://www.shodan.io/
[4] https://github.com/boscutti939/Cowrie_Detect

2 Research Process

This section describes the process in which this study was conducted. The research process (RP) comprises three main phases; sample selection, conduct experiments and observation.

2.1 RP-1: Sample Selection

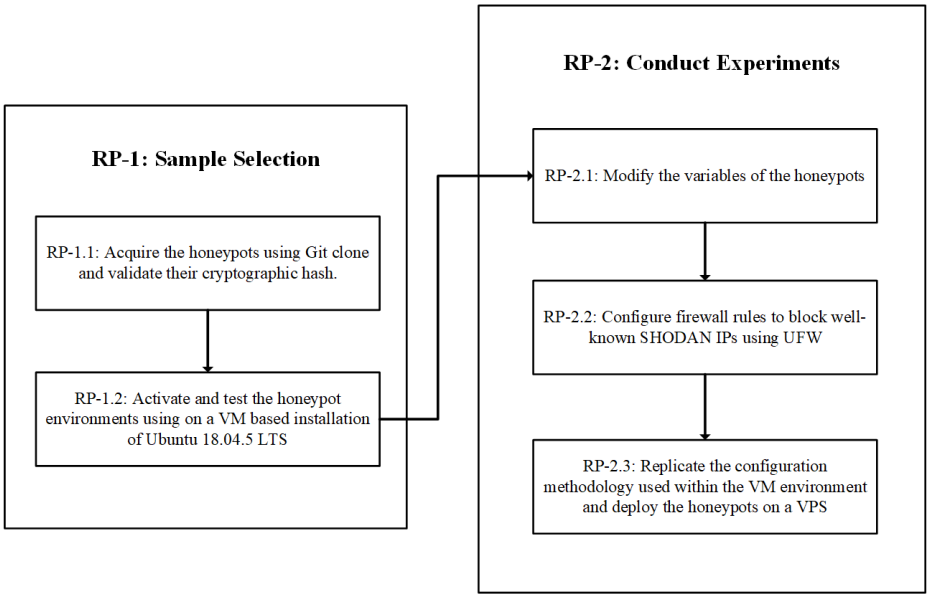

Fig. 1. Sample Selection and Conduct Experiments research phases.

The RP-1 phase consists of two steps (see Figure 1):

i. *RP-1.1:* Acquire the latest stable version of the Cowrie honeypot using the `git clone` command, i.e., Cowrie v2.1.0. After downloading the honeypot files, verify their cryptographic hash to ensure that they were copied with all dependencies and no file corruption.

ii. *RP-1.2:* Activate and test the honeypot environments using virtual machine (VM) based installation of Ubuntu 18.04.5 LTS. A VM environment helps with frequent configurations during the testing phases and provides rollbacks to recover from erroneous designs via snapshots. These configurations can later be replicated (using an automated deployment script) for a configured honeypot deployment on a VPS, as discussed in RP-2.3.

2.2 RP-2: Conduct Experiments

The RP-2 phase comprises of three steps as described (see Figure 1):

i. *RP-2.1:* As seen earlier for Cowrie [3], there is a default configuration file; `cowrie.cfg` can be analyzed and modified using a text editor. Similarly, Cowrie has numerous other files such as `ifconfig.py` (for changing the default MAC address and other base utilities), `userdb.txt` (for managing Cowrie's access credentials) and `honeyfs` files (for managing user groups, CPU information and others). These modifications can be achieved by manually inputting additional scripts, altering variables, enabling (uncommenting), or disabling (commenting) features within the application, or using automated scripts to generate a random set of values for deployment. In this scenario, the `obscurer.py`[5] script was used to automatically configure the Cowrie honeypot artefacts, as shown in Table 1. This script specifically targets the `honeyfs` file system of the Cowrie honeypot – the fake filesystem whose contents mimic an existing design. The script uniquely modifies the artefacts of the honeypot each time it is executed. Furthermore, to reduce the Cowrie honeypot's identification probability, a custom introductory banner was created, and the interactive session timeout was increased from 180 to 600 seconds. Additionally, the `txtcmds` file was configured to display fake outputs for commands such as `ip a`, which otherwise returned empty strings for default Cowrie versions.

Table 1. Configurations employed by the `obscurer.py` script towards Cowrie artefacts.

Function Name	Usage	File Location
`ifconfig_py()`	Creates and modifies the address resolution protocol (ARP) table. Also, it rewrites the `ifconfig` command in Cowrie to show legitimate MAC addresses.	`/src/cowrie/commands/ifconfig.py`
`version_uname()`	Alters the OS version username in `honeyfs` to a different OS version username.	`/honeyfs/proc/version`
`meminfo_py()`	Alters the memory information using a random base value.	`/honeyfs/proc/meminfo.py`
`mounts()`	Alters the name of a few mounted drives for the Cowrie instance.	`/honeyfs/proc/mounts`
`cpuinfo()`	Edits information about the CPU model, clock speed, flags, and cache size for the Cowrie instance.	`/honeyfs/proc/cpuinfo`
`group()`	Deletes the default user and group "`phil`" and adds some random	`/honeyfs/etc/group6`

[5] https://github.com/boscutti939/obscurer

	usernames and groups pulled from the script.	
passwd()	Deletes the default user "phil" and adds some random users pulled from the script.	/honeyfs/etc/passwd
shadow()	Deletes the default user "phil" and adds some random users while generating salted hashed passwords for the users.	/honeyfs/etc/shadow
cowrie_cfg()	Configures the Cowrie cowrie.cfg configuration file (the duplicate copy of the original Cowrie cowrie.cfg.dist file) to change the hostname, OS version name, SSH version, IP address, and other variables.	/etc/cowrie.cfg
hosts(cowrie_install_dir)	Replaces the default/common host "nas3" with another hostname.	/honeyfs/etc/hosts
hostname_py(cowrie_install_dir)	Changes the hostname file to a different hostname (name of the honeypot or server).	/honeyfs/etc/hostname.py
issue(cowrie_install_dir)	Changes the OS issue from the issue file in the honeyfs to a different OS issue.	/honeyfs/etc/issue
userdb(cowrie_install_dir)	Creates and modifies the userdb.txt file and replaces whatever is in it with its unique users and passwords.	/etc/userdb.txt
fs_pickle(cowrie_install_dir)	Executes the createfs command within the Cowrie repository to recreate the honeypot filesystem (This function is executed last by the obscurer.py script)	/share/cowrie/fs.pickle

ii. *RP-2.2:* Cybercriminals use scanning services such as Shodan and Censys for identifying whether they are connecting to legitimate servers or services. According to Chen, Lian [11], security specialists can create an IP blacklist as a predominant solution to block or inhibit scanners. Numerous open sources publish and update the various scanners' IP list. These IP addresses can be added to a firewall's blocklist for preventing automated scanning services up to a certain extent. However, it is essential to remember that an IP list encompassing a complete list of all the automated scanners cannot be established. A list of the well-known automated scanners and their subnets were identified. The developed blacklist was created by referring to GitHub, SANS Institute and Reddit, and the active honeypots themselves. To increase

the honeypots deceptive potential against such scanners, the IP addresses were blocked by using the `ufw` firewall.

iii. *RP-2.3:* Once the configurations are complete, the next step of experimentation phase is to check if there is any increase in the honeypots' deceptive potential. This validation was achieved by replicating the actions seen in RP-2.1 and RP-2.2 by deploying three honeypots; one with default configurations and two using an automated Cowrie deployment script created by using the author's designs, to monitor and validate the deceptive effect of the default deployments versus those of the automatic deployment. These deployments were administered on a monthly cycle, as seen in Table 2.

Table 2. Cowrie Deployment Cycle.

Honeypot	**Deployment Type**	**Deployment Period**	**Credentials**
Cowrie A	Default	1st November 2020 – 29th November 2020	Default credentials
Cowrie B	Configured	30th November 2020 – 28th December 2020	Username*:* tech Password*:* enable
Cowrie C	Configured	29th December 2020 – 26th January 2021	Username: root Password: nproc

2.3 RP-3: Observation

The default honeypots were observed against the configured deployments to study the distinction between the initial and the modified deceptiveness displayed by the honeypots. The following variables can analyze this distinction:

i. *Emulated service activity (SI):* The number of times there was activity with an emulated service.

ii. *IP connections (IP):* the total sum of distinctive machines that attempted to gain access to the Cowrie honeypot.

iii. *Connection to deceptive ports (CDP):* the number of times a cybercriminal connected to deceptive TCP/UDP ports.

iv. *Brute force count (BF):* the highest number of times the same attacker tried to connect to the honeypot instance.

The correlations between these variables were established employing statistical tables in the results section of this research. These values are collected by Cowrie's log files and stored within their respective directories. Splunk was used to visualize the information and study cybercriminal behavior in the configured environment versus the default environment. In cases of a slight increase in the deceptive ability of the honeypots, RP-2 was conducted repeatedly until a satisfied deception capability was

achieved (such as creating supplementary firewall rules). Additional arrangements to test the increase of the deceptive potential was achieved by scanning Cowrie with NMAP 7.80, Shodan and the `cowrie_detect.py` script using a Kali Linux (version) instance.

3 Results and Analysis

In this section, the results of each honeypot are shown and analyzed. These results include the NMAP and Shodan scans, log data collected from Splunk and the output from the `cowrie_detect.py` script. Please note that some data must be hidden due to security purposes, but conclusions can still be drawn.

3.1 NMAP Scan Analysis

NMAP scans were conducted using the command:

```
nmap -sV -Pn -p- [IP_address]
```

Table 3. NMAP scan output for Cowrie A.

Port	State	Service	Version
22/tcp	Open	SSH	`OpenSSH (7.6p1 Ubuntu-4ubuntu0.3)`
2222/tcp	Open	SSH	`OpenSSH (6.0p1 Debian 4+deb7u2)`

Table 4. NMAP scan output for Cowrie B.

Port	State	Service	Version
22/tcp	Open	SSH	`OpenSSH (7.6p1 Ubuntu- 4ubuntu0.3)`
53/tcp	Closed	Domain	N/A
80/tcp	Closed	HTTP	N/A
47808/tcp	Closed	BACnet	N/A
10000/tcp	Closed	Sent-sensor-mgmt	N/A

Where `sV` is used for service detection, `Pn` is used to suppress pings during scans to detect whether a host is up, and `p` is used to specify a port range. In this case, `-p-` scans all ports between 1-65535. The NMAP scan results are shown in Table 3 (for Cowrie A) and Table 4 (for Cowrie B and Cowrie C).

The NMAP scan for Cowrie A revealed that the honeypot was listening on SSH port 22 operating on `OpenSSH (7.6p1 Ubuntu-4ubuntu0.3)` and port 2222 using `OpenSSH (6.0p1 Debian 4+deb7u2)`. A majority of default Cowrie instances operate on two SSH ports; port 2222 is the default port for the Cowrie honeypot and OpenSSH (6.0p1 Debian 4+deb7u2) standard SSH string as seen in the

`cowrie.cfg` config file. These comparisons are solid indicators for a cybercriminal to notice that the machine they are trying to compromise might be a honeypot.

In contrast, Cowrie B and Cowrie C's results displayed that the honeypots allowed incoming connections on a single port, i.e., SSH port 22 using `OpenSSH (7.6p1 Ubuntu-4ubuntu0.3)`. It further detected four closed ports, as seen in Table 4, which is not the case for standard deployments. Furthermore, Cowrie B and Cowrie C did not produce any suspicious honeypot indicators such as standard operating ports, default SSH string versions and static MAC addresses. Therefore, having no default values of being a honeypot instance. The Splunk Log Analysis can prove this increased deceptiveness from NMAP scans (see section 3.3), where Cowrie B and Cowrie C had a more significant percentage of SI, IP, CDP, and BF as equated to Cowrie A.

3.2 Shodan Scan Analysis

The Shodan scans revealed the same information about the honeypot instances as described in the NMAP scan. It showed that Cowrie A was listening on two open ports, port 22 and port 2222, while Cowrie B and Cowrie C were listening on port 22. It also displayed the default SSH algorithms used by Cowrie A (see `cowrie.cfg`) instead of those used by the other two honeypot instances (see Table 5). The default SSH strings are indicators for a cybercriminal to detect a Cowrie instance in conjunction with the discovered open ports. This indicator is not the case for Cowrie B and Cowrie C because they use configured SSH algorithms on port 22.

Table 5. SSH algorithms used by the Cowrie honeypot instances.

SSH Algorithm	Cowrie A	Cowrie B & Cowrie C
KEX Algorithms	`curve25519-sha256, curve25519-sha256@libssh.org, ecdh-sha2-nistp256, ecdh-sha2-nistp384, ecdh-sha2-nistp521, diffie-hellman-group-exchange-sha256 , diffie-hellman-group-exchange-sha1, diffie-hellman-group14-sha1`	`curve25519-sha256, curve25519-sha256@libssh.org, ecdh-sha2-nistp256, ecdh-sha2-nistp384, ecdh-sha2-nistp521, diffie-hellman-group-exchange-sha256, diffie-hellman-group16-sha512, diffie-hellman-group18-sha512, diffie-hellman-group14-sha256, diffie-hellman-group14-sha1`
Server Host Key Algorithms	`ssh-rsa, ssh-dss`	`ssh-rsa, rsa-sha2-512, rsa-sha2-256,`

		ecdsa-sha2-nistp256, ssh-ed25519,
Encryption Algorithms	aes128-ctr, aes192-ctr, aes256-ctr, aes256-cbc, aes192-cbc, aes128-cbc, 3des-cbc, blowfish-cbc, cast128-cbc	chacha20-poly1305@openssh.com, aes128-ctr, aes192-ctr, aes256-ctr, aes128-gcm@openssh.com, aes256-gcm@openssh.com
MAC Algorithms	hmac-sha2-512 hmac-sha2-384 hmac-sha2-56 hmac-sha1 hmac-md5	umac-64-etm@openssh.com, umac-128-etm@openssh.com, hmac-sha2-256-etm@openssh.com, hmac-sha2-512-etm@openssh.com, hmac-sha1-etm@openssh.com, umac-64@openssh.com, umac-128@openssh.com, hmac-sha2-256, hmac-sha2-512, hmac-sha1
Compression Algorithms	zlib@openssh.com zlib none	None, zlib@openssh.com

3.3 Splunk Analysis

Splunk was used to visualize data collected from the Cowrie logs. This information has been aggregated into three datasets based on a:

i. *24-hour deployment*: As seen in Table 6, Cowrie A detected 998 instances of service emulation activity. Cowrie B did not detect any emulation activity cases because not even a single attacker gained access to the honeypot instance even after recording 4,601 brute force attempts from a single IP. The honeypot had emulation inactivity even after 7 and 28 days of being deployed. In contrast, Cowrie C attained 9,921 SI's, an 894.08% increase than Cowrie A. It also detected 243 distinct IP connections, a 38.85% and a 5.65% enhancement compared to Cowrie A and Cowrie B honeypots. Cowrie A established connections on port 2222 receiving 1,198 requests in terms of CDP, whereas Cowrie B connected to port 22 and port 443, receiving 10,549 and 3.922 probes, respectively. Cowrie C allowed connections on 16

deceptive ports. On its SSH port, it received 18,598 searches, a 1452.42% and 76.30% increase compared to the other honeypots. It also recorded the highest number of BF occurrences, 18,650 attempts.

Table 6. Information collected from the Cowrie honeypots for a 24-hour deployment.

Honeypot	**SI**	**IP**	**CDP**		**BF**
			Port	**Count**	
Cowrie A	998	175	2222	1,198	3,976
Cowrie B	0	230	22 443	10,549 3,922	4,601
Cowrie C	9,921	243	443 22 80 993 25000 25 5555 143 465 587 995 2525 43 26 4013 2002	31,039 18,598 12,391 6,040 1,148 1,043 412 287 224 133 46 23 10 10 2 2	18,650

ii. *7-day deployment*: In Table 7, the statistics show that the configured honeypots had many cybercriminals connected to deceptive ports. Cowrie A recorded 7,210 connections, Cowrie B had 79,492, and Cowrie C recorded 481,204 connections. Cowrie B verified a 1002.52% increase and, Cowrie C logged a 6,574.12% augmentation in terms of CDP compared to Cowrie A. The configured honeypots also showed superior deceptiveness in terms of SI (excluding Cowrie B), IP, and BF compared to Cowrie A.

Table 7. Information collected from the Cowrie honeypots for a 7-day deployment.

Honeypot	**SI**	**IP**	**CDP**		**BF**
			Port	**Count**	
Cowrie A	10,347	1,198	2222	7,210	15,992
Cowrie B	0	1,589	22 443	55,722 23,770	18,541
Cowrie C	61,823	1,675	443 22 80 993	169,499 129,893 85,451 72,946	56,081

			25000	6,645	
			25	6,583	
			465	2,690	
			5555	2,663	
			143	2,306	
			587	1,574	
			995	470	
			3724	144	
			2525	123	
			26	90	
			43	70	
			110	40	
			8000	12	
			4013	2	
			2002	2	
			27018	1	

Table 8. Information collected from the Cowrie honeypots for a 28-day deployment.

Honeypot	**SI**	**IP**	**CDP**		**BF**
			Port	**Count**	
Cowrie A	51,644	4,972	2222	49,340	67,508
Cowrie B	0	8,201	22	225,096	78,328
			443	108,084	
Cowrie C	258,332	11,653	22	642,732	239,802
			443	588,977	
			80	334,188	
			993	268,812	
			25000	32,680	
			25	31,221	
			465	13,985	
			5555	12,370	
			143	12,321	
			587	8,249	
			995	2,463	
			3724	932	
			2525	346	
			26	231	
			43	190	
			110	120	
			8000	54	
			4013	13	
			2002	8	
			32400	8	
			27018	6	

iii. *28-day deployment*: Table 8 shows similar results to Table 6 and Table 7, i.e., the configured honeypots had better deceptive indicators in terms of SI, IP, CDP, and BF. Cowrie A displayed 51,644 instances of emulation activity, received connections from 4,972 distinct IP addresses, allowed 49,340 connections to its SSH port and recorded 67,508 brute force attacks. On the other hand, Cowrie B detected a 100% decrease in SI activity, a 64.94% increase in IP connections, allowed 356.21% more deceptive connections compared to Cowrie A on the SSH port, and finally, a 16.02% rise in brute force attempts. Similarly, Cowrie C detected a 400.22% increase in emulation activity, a 134.37% rise in the number of different IP connections, connections to 21 deceptive ports and a 255.22% surge in the number of brute force attempts originating from a particular IP with 239,802 recorded attempts.

3.4 Cowrie_detect.py Detection Analysis

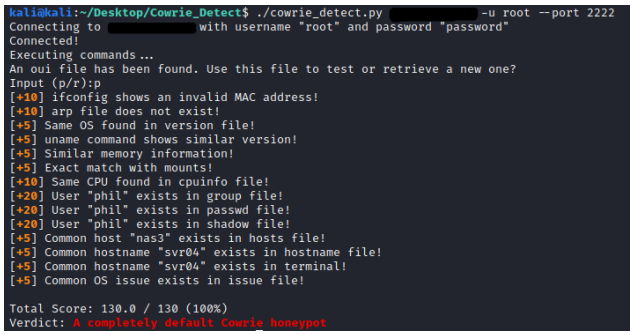

Fig. 2. `cowrie_detect.py` scan on Cowrie A

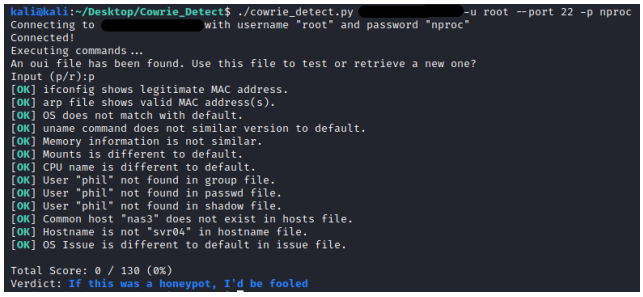

Fig. 3. `cowrie_detect.py` scan on Cowrie B and Cowrie C

Figure 2 shows that the automated script, `cowrie_detect.py`, detected Cowrie A as a default honeypot instance with a 100% certainty. The script's leading indicators to distinguish the default Cowrie instance were the existence of the standard user "`phil`" in the group, password and shadow files, and the presence of a static/invalid MAC address and the absence of an ARP file. Secondary detection indicators are the

occurrences of an unconfigured `cowrie.cfg` file; default memory and CPU information and a pre-existing OS version used by the Cowrie host and the standard hosts "`nas3`" and "`svr04`". In comparison, Figure 3 displays the output of the `cowrie_detect.py` when executed against Cowrie B and Cowrie C. The script could not find any default users, a static MAC address, default CPU data, standard memory information or default configured hostnames. Therefore, it can be verified that the revised versions of Cowrie are 100% more deceptive than the standard versions when scanned using the `cowrie_detect.py` script.

3.5 Miscellaneous Observations

According to the command line results, the most entered commands included `uname -s -v -n -r`, `cat /proc/cpuinfo | grep name | wc -l` and `cat /proc/cpuinfo | grep name | head -n 1 | awk '{print $4,$5,$6,$7,$8,$9;}'`. For the `uname` command, Cowrie A returns the default string, and for the `cpuinfo` commands, it produced the output: `Intel(R) Core(TM)2 Duo E8200 @` while the modified versions generated: `Intel(R) Core(TM) I7-2960XM CPU @ 2.70GHz`. These results show that the command fails on all default versions of the Cowrie instance. It is a powerful indicator for a cybercriminal to exit the environment upon connecting to the honeypot server immediately. Therefore, it was detected that 81% of attackers exited the honeypot session after executing the `uname` command in Cowrie A and 87% of attackers exited the interactive shell after entering the `cpuinfo` command – thus, verifying that on average, IP addresses stayed for 112.36 seconds when connected to Cowrie A. However, commands entered in the configured instances produced a legitimate string with unique information about the shell and CPU, unlike Cowrie A. Finally, Cowrie C showed that attackers executed 65% more commands and stayed within the interactive shell an average of 537.82 seconds after using the `uname` command before exiting the environment.

4 Conclusion

The results of the honeypot deployments show the effectiveness of having configured deployments over the default counterparts. Configured Cowrie honeypots provided an improvement in the information collected from cybercriminals, such as an increase in distinct IP connections, the types of commands being executed, files downloaded and VirusTotal submissions. These instances also displayed the number of brute force attempts and the top username and password combinations used. Cowrie B did not detect any SI activity because attackers could not access its interactive shell due to a more robust username/password combination. The most common login credentials in Cowrie A and Cowrie B's case were recorded to be "`root`" and "`nproc`". Therefore, Cowrie C was configured with this weaker username/password combination, although the automated script uniquely modified its other artefacts. In future, the obscurer script will be further modified, or a new script will be forked to include modifications

to Cowrie's `txtcmds` and `fs.pickle` artefacts to provide a higher simulation in terms of command outputs. The script will also be configured to have several unique new users, hostnames, groups, file shares, hard drive sizes, mounts, CPU and RAM information, OS versions, IP addresses, MAC addresses and SSH versions. Further testings and analysis will also be carried out to show that the script does not configure the honeypot in some deterministic manner which scanning tools can soon automate.

5 Acknowledgements

The work has been supported by the Cyber Security Research Centre, whose activities are partially funded by the Australian Government's Cooperative Research Centres program.

References

1. Cabral, W.Z., Valli, C., Sikos, L.F., Wakeling, S.G.: Analysis of Conpot and its BACnet Features for Cyber-Deception. In: Proceedings of the 19th International Conference on Security & Management. Springer (2020).
2. Morishita, S., Hoizumi, T., Ueno, W., Tanabe, R., Gañán, C., van Eeten, M. J., Yoshioka, K., Matsumoto, T.: Detect me if you… oh wait. An internet-wide view of self-revealing honeypots. 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM). pp. 134-143. IEEE (2019).
3. Cabral, W.Z., Valli, C., Sikos, L.F., Wakeling, S.G.: Review and Analysis of Cowrie Artefacts and their Potential to be used Deceptively. In: Proceedings of the 6th Annual Conference on Computational Science and Computational Intelligence. pp. 166–171. IEEE (2019). doi:10.1109/CSCI49370.2019.00035
4. Zhang, F., Zhou, S., Qin, Z., Liu, J.: Honeypot: a supplemented active defense system for network security. In: Proceeding of the 4th International Conference on Parallel and Distributed Computing, Applications and Technologies. pp. 231-235. IEEE (2003). doi: 10.1109/PDCAT.2003.1236295
5. Nicomette, V., Kaâniche, M., Alata, E., Herrb, M.: Set-up and deployment of a high-interaction honeypot: experiment and lessons learned. Journal in Computer Virology. 7, 143-157 (2010). doi: 10.1007/s11416-010-0144-2
6. Chen, Y., Lian, X., Yu, D., Lv, S., Hao, S., Ma, Y.: Exploring Shodan From the Perspective of Industrial Control Systems. IEEE Access. 8, 75359-75369 (2020). doi: 10.1109/GIIS.2018.8635603
7. Vetterl, A., Clayton, R., Walden, I.: Counting outdated honeypots: Legal and useful. 2019 IEEE Security and Privacy Workshops (SPW). pp. 224-229. IEEE (2019). doi: 10.1109/SPW.2019.00049
8. Oosterhof, M.: Cowrie SSH and Telnet Honeypot: GitHub (2020), https://github.com/cowrie/cowrie
9. Lyon, G.F.: Nmap - Network Mapper (2021), https://nmap.org/
10. Matherly, J.: Shodan (2021), https://www.shodan.io/
11. Chen, Y., Lian, X., Yu, D., Lv, S., Hao, S., & Ma, Y.: Exploring Shodan From the Perspective of Industrial Control Systems. Exploring Shodan From the Perspective of Indus-

trial Control Systems. IEEE Access. 8, 75359-75369 (2020). doi: 10.1109/ACCESS.2020.2988691