Enhancing SSH/Telnet Honeypot for Attack Classification and Malware Research

Michele Castellaneta[1,†], Abraham Gebrehiwot[1,†], Filippo Maria Lauria[1,†] and Claudio Porta[1,*,†]

[1] Istituto di Informatica e Telematica - CNR , via G. Moruzzi 1, Pisa, Italy

Abstract

Nowadays, honeypots are widely used for detecting and analyzing attack activity and behaviors, including login attempts, command executions, or code injection. The ability to collect malware from attackers depends on how effectively the honeypot replicates real devices and whether the attack is targeted at a specific device model or class or is generic. This work improves the Cowrie SSH/Telnet honeypot's effectiveness to enhance data quality and support advanced attack analysis. We implemented a logging system for real-time log collection, normalization, and visualization. Despite high attack volume, many sessions ended early due to bots detecting the honeypot. To mitigate detection, we developed a modified honeypot replica featuring system file changes and access controls using realistic username and password lists. Multiple configurations were tested to assess their effect on attacker behavior. Our results highlight the challenges of evasion by automated attackers but demonstrate that improved honeypot realism leads to more meaningful data for cybersecurity research, enabling the filtering and classification of attacks targeting specific devices or providing a starting dataset for further refinement using machine learning techniques. Future work will focus on further enhancing honeypot stealth and customization and expanding malware analysis capabilities.

Keywords

Cybersecurity, malware detection, SSH and Telnet honeypot

1. Introduction

Honeypots are useful tools that can be employed for multiple purposes, such as detecting attacks against information systems, studying the attack techniques used to adopt appropriate countermeasures, discovering new malware variants, improving network defenses, and collecting valuable data related to the evolution of attack techniques [8]. Modern networks constitute a highly complex ecosystem composed of a wide variety of interconnected and heterogeneous devices. This complexity gives rise to numerous attack vectors, which must be carefully analyzed to enable effective mitigation and prevention strategies. In this context, our work concentrates on examining attacks aimed at the two primary network protocols employed for remote device management: Telnet and SSH.

This kind of analysis can be conducted by deploying a honeypot exposed to the Internet via a public IP address, simulating a system accessible through Telnet or SSH. In this way, attacks and attacker interactions with the system can be recorded. One widely used open-source application designed for this purpose is Cowrie [7], a medium-interaction SSH/Telnet honeypot that, in its emulated shell mode, uses Python to mimic a real system. This allows it to log login attempts and capture key information about the nature of the attacks. However, this simulation approach has two main challenges: first, the honeypot must be difficult for attackers to detect, as otherwise they tend to abandon the attack or compromise the reliability of the collected data [1]; second, managing the large volume of captured information can complicate data analysis and interpretation. The volume of data depends both on the realism level of the honeypot and on the type and frequency of attacks received over time. One of the main challenges is the risk of detection by attackers, who may identify the honeypot through specific system or network clues.

Joint National Conference on Cybersecurity (ITASEC & SERICS 2026), February 09-13, 2026, Cagliari, IT

* Corresponding author.

† These authors contributed equally.

michele.castellaneta@iit.cnr.it (M. Castellaneta); abraham.gebrehiwot@iit.cnr.it (A. Gebrehiwot); filippo.lauria@iit.cnr.it (F. M. Lauria); claudio.porta@iit.cnr.it (C. Porta)

Once uncovered, the honeypot loses its effectiveness, as the attacker's behavior no longer accurately reflects real-world attacks [2].

This paper proposes possible solutions to mitigate these issues by applying obfuscation techniques to Cowrie [7] and carefully selecting authentication credentials. The goal is to make it more difficult for attackers, even those using automated detection tools or techniques, to recognize that they are interacting with a honeypot. A fully open-source and customizable solution for the analysis, processing, and exploration of the collected data is also presented.
The paper then describes the testing phases conducted on the implemented architecture to demonstrate its effectiveness, along with an analysis of the obtained results. Two aspects are particularly noteworthy: the integration of an additional feature to identify attack types and classify devices belonging to the same botnet based on common properties and similarity criteria, and the preliminary creation of a dataset containing commands used during attacks. This dataset supports the structured classification of attacks and provides real-world data for future work involving machine learning algorithms trained on filtered, high-quality information.
The paper also includes a section on related work, highlighting similarities and differences with our approach, followed by concluding remarks and directions for future research.

2. Related Works

As effectively summarized in [3], the use of honeypots has increased significantly in recent years, driven by the widespread adoption of IoT devices and distributed computing paradigms, which have introduced new and substantial cybersecurity challenges. For instance, [4] presents tools for deploying honeypots and a system for detecting attacks on TCP and UDP ports, focusing primarily on identifying attackers' IP addresses and their geographical origin. However, aspects related to honeypot deception capabilities and advanced data management remain unaddressed, raising concerns about the actual reliability of the collected data and the potential presence of many false positives. Other studies, such as [9], leverage large language models for the analysis of SSH and HTTP traffic logs generated by honeypots integrated with Elasticsearch, achieving significant results for certain classes of malware, and successfully overcoming obfuscation techniques employed by attackers. Nevertheless, in this case as well, honeypots are deployed using default configurations, which leads to the loss of attacker activities once the honeypot is identified as such and does not address the management of potential false positives. Numerous works have focused on the development of deception techniques [10, 15], including the simulation of IoT devices [14], alongside parallel research efforts on honeypot fingerprinting [2, 5].

The objective is always the same: to develop honeypots that are difficult for attackers to detect. A recent study closely related to our work, focusing on the analysis of SSH/Telnet attacks [6], enhances the interaction and deception capabilities of the Cowrie honeypot [7] by improving the authentication mechanism and redesigning the simulated filesystem to support persistent content, thereby allowing multiple users to share a single Cowrie instance. However, the conducted research does not account for attackers who generate credentials randomly, which may lead to the detection of the honeypot. Furthermore, it does not include filtering and classification mechanisms based on attack types, such as distinguishing attacks targeting specific devices, where attackers access the system via Telnet or SSH but, failing to find the intended target, do not issue any commands.

Our work introduces minor modifications to the authentication configuration and the simulated filesystem, which are sufficient to improve the quality of the collected data compared to the default setup, and to enable session categorization based on attack fingerprint analysis and access credentials. Another contribution that partially overlaps with our approach is presented in [11], which compares a Cowrie deployment using the standard configuration with a modified version designed to be indistinguishable to attackers through the addition of new commands and configuration changes. That study provides several statistics primarily related to executed commands, ultimately concluding that the modified honeypot can collect a larger amount of information than the default configuration.

In our work, modifications to the filesystem and configurations, following the approach described in [12], were implemented by editing and using two scripts, *cowrie_detect.py* and

obscurer.py. The first is specifically designed to detect Cowrie instances, while the second is designed to remove most of the default values from a Cowrie installation. Additionally, we restrict access to a limited set of default or commonly used credentials in attacks and extend the analysis to the session level, enabling us to identify sessions that successfully conclude without generating any commands. This approach allows us to associate sessions with the same botnet and to differentiate attacks targeting specific devices from those aimed at generic SSH servers. Using a method like [13], we also generated a command dataset to analyze attacker behavior based not only on shell activity but also on network traffic.

3. Goal and Setup Environment

Our research aims to identify dictionary attacks and unauthorized access attempts, analyze commands executed by attackers and any files downloaded after accessing the system, and generally collect as much useful information as possible about the operations carried out during the attack.

3.1. Honeypot Deployment

The first phase involved the creation of a virtual machine running a Unix-based operating system, on which, alongside standard system libraries, an instance of the Cowrie honeypot was installed and configured to serve emulated Telnet and SSH services. This virtual machine was then connected to the internet with a public IP address, thus simulating an inadequately secured network device.

3.1.1. Cowrie

As briefly mentioned earlier, Cowrie is a modular, medium-interaction open-source honeypot written in Python that supports SSH and Telnet protocols. It can be configured to run in different modes: as an emulated shell (default), simulating a fake file system resembling a standard Linux installation and providing shell interaction with common system commands; as a proxy, forwarding SSH and Telnet traffic to another system or managing a pool of QEMU emulated servers; or using an LLM backend (experimental), leveraging large language models to dynamically generate realistic shell responses [7]. It logs attacker interactions with the shell, including username and password combinations used to access the system, IP addresses, SSH and Telnet client versions, file uploads and downloads, executed commands, and more. By default, this information is stored in *.log* and *.json* formats, intended for manual and automated analysis respectively. For our work, we used the emulated shell mode, as it best suited our purposes.

3.1.2. Setup for Data Processing and Visualization

However, the large volume of gathered data and the continuous updates by the application make direct log analysis and interpretation challenging. To simplify this task, similarly to previous works [6, 11], Cowrie was integrated with tools for log analysis, storage, and graphical rendering of key information. The implemented architecture is shown in Figure 1.

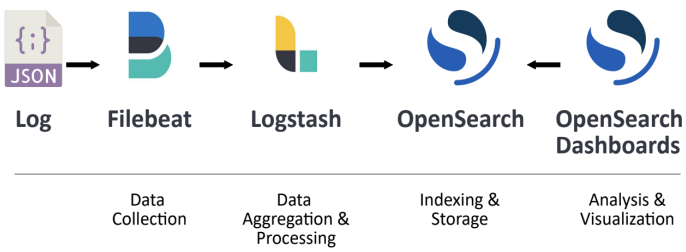

Figure 1: Architecture for data processing and visualization

- Filebeat[1] monitors and collects logs in real time, forwarding them to one or more configurable outputs.
- Logstash[2] unifies and normalizes data from multiple sources. At this stage, log data can be enriched, filtered, and transformed to extract more relevant information.
- OpenSearch[3] is a distributed search and analytics engine that enables extensive querying of data, with support for filtering, aggregation, and sorting.
- OpenSearch Dashboards[4] is a data visualization tool that allows users to explore, analyze, and understand data in real time.

Each tool was installed, configured, and tested individually. The architecture integration was then verified by confirming that Cowrie logs were stored and displayed in real time on OpenSearch Dashboards. During the verification phase, test attacks were performed against the honeypot to confirm the accuracy of logged actions and timestamps.

3.2. Data Gathering

Once the testing phase was complete, the container was exposed to the Internet and data collection began. The initial metrics gathered during this phase were as follows:

- top 10 most frequently used usernames in access attempts.
- top 10 most frequently used passwords.
- top 10 IP addresses with the most activity on the honeypot.
- percentage breakdown of operations by destination port.
- number of unique source ports from which attacks originated.
- top 10 most frequently used commands by attackers.
- total number of attack sessions.
- number of sessions with at least one operation, and their percentage relative to total sessions.
- session duration estimates (minimum, average, and maximum).
- SHA256 checksums of the most downloaded files via HTTP (typically using wget and curl).
- SHA256 checksums of the most uploaded files via FTP.
- SHA256 checksums of files with the most failed download attempts.
- number of unique interactions, sorted by type.
- percentage distribution of attacker SSH client versions.

Figure 2 shows part of the data displayed graphically by the OpenSearch dashboard interface.

The HASSH library was also integrated into the infrastructure. HASSH[5] is a network fingerprinting standard used to identify specific SSH client and server implementations. Fingerprints can be easily stored, searched, and shared as compact MD5 hashes. Using HASSH, we computed fingerprint hashes of detected SSH clients, providing an additional parameter for malware tracking.

[1] Filebeat - https://www.elastic.co/beats/filebeat
[2] Logstash - https://www.elastic.co/logstash
[3] OpenSearch - https://opensearch.org/
[4] OpenSearch Dashboards - https://docs.opensearch.org/latest/dashboards/
[5] HASSH - https://github.com/salesforce/hassh

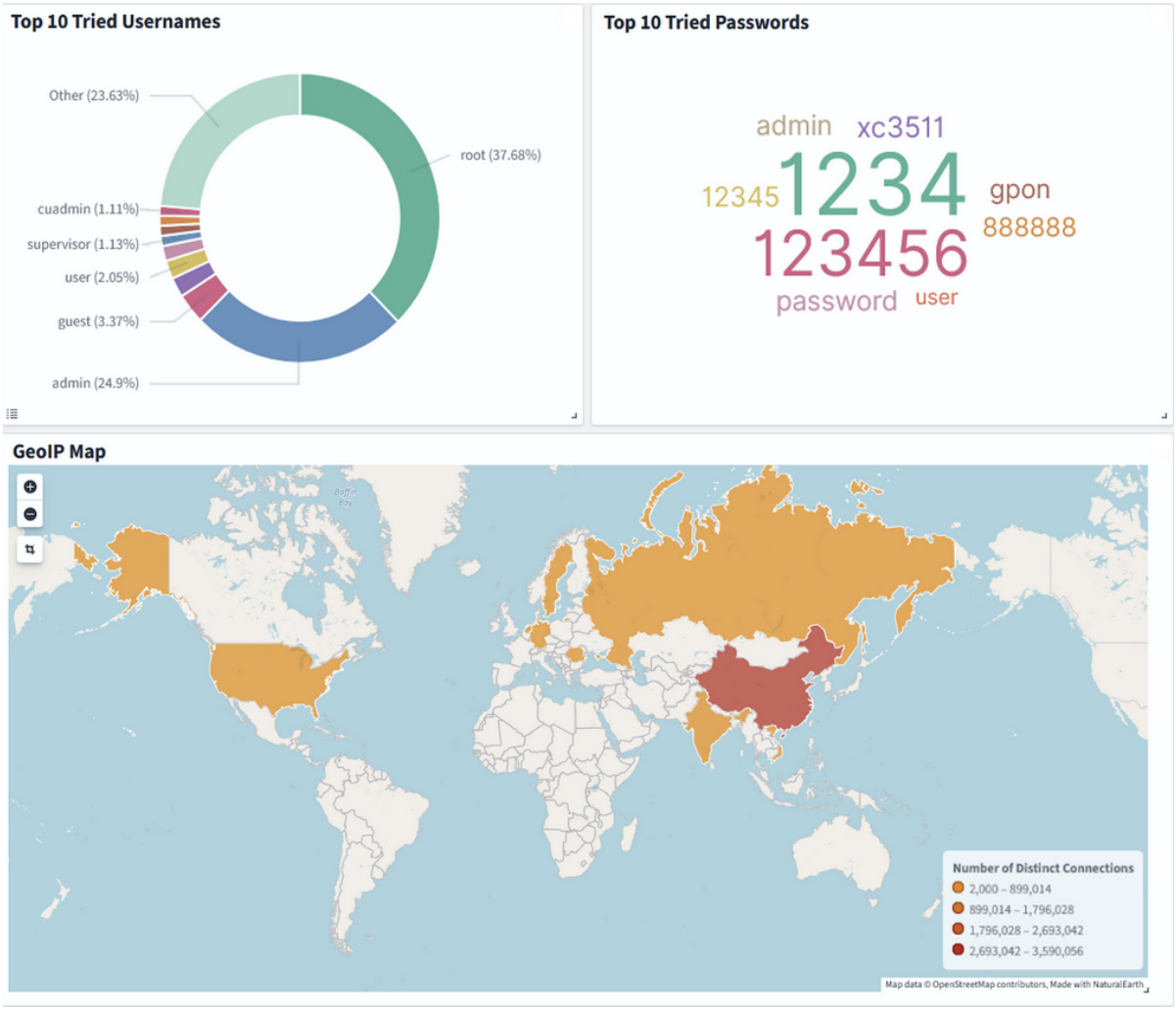

Figure 2: Graphical elements of the OpenSearch dashboard interface

Finally, based on SSH client type, a percentage estimate was computed to distinguish connections from bots versus real users, confirming that most attacks, as session duration data also suggested, were carried out by bots.

3.3. Initial Results

Initial analysis revealed that data collected from the honeypot in its default configuration was insufficient for accurate attack classification. Despite numerous daily connection attempts, most terminated quickly without executing any commands, even though the connection had been successfully established. We hypothesize this is due to at least three main factors:

1. botnets capable of recognizing they have landed on a Cowrie instance. An example of a script attackers can use to detect the honeypot before proceeding with an attack, as mentioned in [12], is *cowrie_detect.py*[6]. It works by comparing file contents on the target machine against default files in Cowrie distributions. After modifying the script to support the latest version of Cowrie (2.8.1), we tested it against our honeypot (with default configuration), successfully confirming detection.

[6] cowrie_detect - https://github.com/boscutti939/Cowrie_Detect.

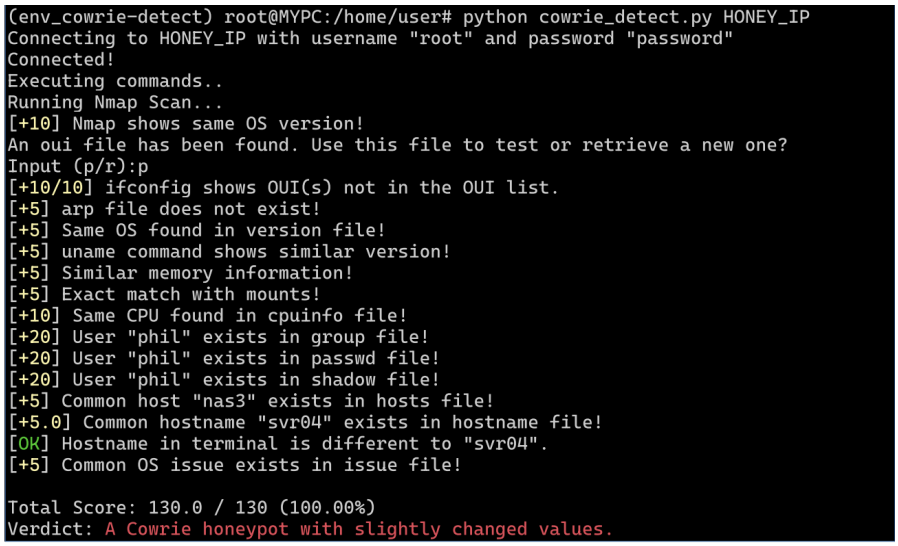

Figure 3: Output of cowrie_detect.py detecting a default Cowrie installation

2. Cowrie enforces a default access policy that permits logins also to a limited set of default usernames, namely *root* and *phil*. This behavior can inadvertently disclose the presence of the honeypot, particularly when authentication attempts using the username phil succeed. We replaced the default access policy with another unmanaged and permissive one that accepts arbitrary credentials, but a different issue emerges: attackers authenticating with randomly generated usernames and passwords can detect the honeypot due to its unrealistic authentication behavior.
3. attacks are highly diverse. Analysis of access credentials and executed commands reveals that some target specific devices using default credentials associated with particular device classes, while others attempt to access generic systems using weak or commonly used credentials.

4. Improvements

4.1. Cowrie Obfuscation

Based on the hypotheses derived from the initial results analysis (Section 3.3), extensive work was carried out to make the honeypot less recognizable. Initially, a clone of the system was created to apply modifications while preserving the original for future comparisons. The *obscurer.py*[7] script was then modified and used to automatically change Cowrie's default values. Table 1 summarizes the configuration changes applied to Cowrie for obfuscation purposes.

[7] Cowrie Honeypot Obscurer - https://github.com/boscutti939/obscurer

Table 1
Summary of configuration changes applied to Cowrie using the *obscurer.py* script

Function	Customization	Involved file on customization
ifconfig_py()	Modified ARP and ifconfig command output	/src/cowrie/commands/ifconfig.py
version_uname()	Customized kernel version	/honeyfs/proc/version
meminfo_py()	Modified system memory information	/honeyfs/proc/meminfo.py
mounts()	Customized mounted devices	/honeyfs/proc/mounts
cpuinfo()	Modified CPU information	/honeyfs/proc/cpuinfo
group()	Modified system groups	/honeyfs/etc/group
passwd()	Modified system users	/honeyfs/etc/passwd
shadow()	Modified authentication credentials	/honeyfs/etc/shadow
cowrie_cfg()	Configured hostname, OS, SSH service	/etc/cowrie.cfg
host (cowrie_install_dir)	Replaced default "nas3" value	/honeyfs/etc/hosts
hostname_py (cowrie_install_dir)	Modified system hostname	/honeyfs/etc/hostname.py
issue (cowrie_install_dir)	Replaced OS issue file	/honeyfs/etc/issue
userdb (cowrie_install_dir)	Created and modified userdb.txt	/etc/userdb.txt
fs_pickle (cowrie_install_dir)	Regenerated simulated filesystem	/share/cowrie/fs.pickle

Following the implementation of these changes on the cloned honeypot, *cowrie_detect.py* was run again, this time classifying the machine as a realistic system.

```
(env_cowrie-detect) root@MYPC:/home/user# python cowrie_detect.py HONEY_CLONE_IP
Connecting to HONEY_CLONE_IP with username "root" and password "password"
Connected!
Executing commands..
Running Nmap Scan...
[OK] Nmap shows different OS version to default.
An oui file has been found. Use this file to test or retrieve a new one?
Input (p/r):p
[OK] ifconfig shows valid MAC address(s).
[OK] arp file shows valid MAC address(s).
[OK] OS does not match with default.
[OK] uname command does not similar version to default.
[OK] Memory information is not similar.
[OK] Mounts is different to default.
[OK] CPU name is different to default.
[OK] User "phil" not found in group file.
[OK] User "phil" not found in passwd file.
[OK] User "phil" not found in shadow file.
[OK] Common host "nas3" does not exist in hosts file.
[OK] Hostname is not "svr04" in hostname file.
[OK] OS Issue is different to default in issue file.
Verdict: Real system.
```

Figure 4: Output of cowrie_detect.py after applying obfuscation changes

4.2. Access Control Configuration

To make the honeypot even more realistic, we implemented a custom access control system, as Cowrie's default distribution allows login with any username and password combination or a small set of predefined users, namely *root* and *phil*. By default, Cowrie uses an userdb.txt file where wildcards can be used to accept any credentials (e.g., *root:x:** grants access to user root with any password, while **:x:** accepts any combination). In particular, to restrict attacker access, we generated credential lists from publicly available sources[8,9], including both commonly used credentials and default credentials for various Internet-connected devices.

The intersection of these lists enabled the identification of 1127 unique default usernames and 97891 unique passwords. The resulting password set was organized by listing default passwords first (1667 entries), followed by commonly used passwords (96224 entries) sorted by frequency of use. These credentials were then used to restrict attacker access and prevent the use of randomized credentials.

5. Testing and Results

To evaluate the effectiveness of the applied changes, five different trials (000, 001, 002, 003, 004) were conducted over a one-month period. Each trial lasted 108 hours:

- trial 000 has no modifications in the default configuration and no access policy implemented, meaning that any username–password combination is accepted.
- trial 001 allows default usernames and any password combination.
- trial 002 allows only the access of the default usernames combined with the first 100 default passwords.
- trial 003 relaxes the password restriction compared with trial 002, allowing the first 500 default passwords in combination with the default usernames.
- trial 004 has been repeated using the same configuration as trial 001 to determine whether the low number of attacks recorded during that period was due to the chosen configuration or to random variability.

Table 2 summarizes the conducted tests and their access policy configuration.

Table 2
Summary of trial configurations and access control settings

Trial ID	Users	Passwords	Start Date	End Date	Duration (hours)
000	any	any	2025-10-15	2025-10-21	108
001	1127 default	any	2025-10-23	2025-10-28	108
002	1127 default	top 100 default	2025-10-30	2025-11-04	108
003	1127 default	top 500 default	2025-11-06	2025-11-11	108
004	1127 default	any	2025-11-26	2025-12-02	108

The analysis of attacker-initiated sessions first considered, for each trial, the total number of observed sessions *(TS)*. We subsequently distinguished between successfully established sessions *(SS)*, in which the attacker gained access to the system, and unsuccessful sessions, in which the attacker failed to obtain access. Among the successfully established sessions, interactions were

[8] SecLists (CIRT default usernames, default-passwords, etc.) - https://github.com/danielmiessler/SecLists
[9] Default Credentials Cheat Sheet - https://github.com/ihebski/DefaultCreds-cheat-sheet

further classified into active sessions *(AS),* in which the attacker executed at least one command, and passive sessions *(PS),* in which the attacker disconnected without performing any action. Table 3 provides an overview of the collected data. The first column shows the trial ID, the second shows total sessions, the third shows successful sessions, and the fourth and fifth columns show active and passive sessions respectively. The sixth column indicates the percentage of successful sessions relative to total sessions, while the last column shows the percentage of active sessions relative to successful sessions.

As mentioned above, active sessions are defined as those where attackers executed commands or attempted to upload or download files, while passive sessions are those with no recorded activity that terminated without generating any information.

Table 3
Session statistics by trial: total, successful, active, and passive sessions with percentages

Trial ID	Total Sessions (TS)	Successful Sessions (SS)	Active Sessions (AS)	Passive Sessions (PS)	% (SS/TS)	% (AS/SS)
000	91276	89988	87766	2222	98,59%	97,53%
001	24270	9743	5807	3936	40,14%	59,60%
002	476623	51313	11845	39468	10,77%	23,08%
003	205469	25503	8575	16928	12,41%	33,62%
004	163759	142159	137507	4652	86,81%	96,73%

In Table 4, the data in the *Login Attempts* and *Successful Logins* columns correspond to absolute event counts per trial. Comparing login numbers with session counts reveals a similar trend. Additionally, as expected, each successful session corresponds to a successful login.

Table 4
Login attempts and successful logins per trial

Trial ID	Logins Attempts (LA)	Successful Logins (SL)	% (SL/LA)
000	91792	89988	98,03%
001	23718	9743	41,08%
002	321418	51313	15,96%
003	110294	25503	23,12%
004	166675	142159	85,29%

5.1. Analysis of Results

Analysis of the data in Table 3 and 4 reveals some notable observations. The overall number of login attempts and total sessions shows considerable variability over time and appears largely uncorrelated with the tested configurations. This is particularly evident when comparing trial 001 and trial 004, which share the same configuration yet exhibit vastly different values. We hypothesize that these variations may be driven by fluctuations in attack campaign activity during the test periods or by other external factors; these hypotheses warrant further investigation in future work.

We also observed sessions in which attackers successfully authenticated but did not execute any commands. One possible explanation is that such behavior is more prevalent in targeted attacks, which are inherently specific to a particular class of devices and may not be effectively captured by Cowrie's generic emulation. In this context, attacks can be broadly classified into targeted attacks, aimed at specific device families, and generic attacks. Since our system emulates a generic Linux-like environment, an attacker performing a targeted attack using device-specific credentials may encounter inconsistencies with the expected target. For instance, discrepancies in the login banner or other passive fingerprinting artifacts may allow the attacker to infer that the system is not a genuine device and subsequently terminate the session without further interaction.

Furthermore, the initial data analysis suggested that leaving the honeypot in its default configuration—without a custom access control policy and additional deception mechanisms, as in trial 000—was more effective in generating active sessions and thus gathering a greater amount of information. However, this hypothesis is partially challenged when the analysis is refined by classifying sessions according to protocol. To verify these assumptions and determine whether the behavior is consistent across both protocols, we separately analyzed SSH and Telnet traffic, as reported in Tables 5 and 6.

Table 5
SSH sessions per trial

	SSH					
Trial ID	**Total Sessions (TS)**	**Successful Sessions (SS)**	**Active Sessions (AS)**	**Passive Sessions (PS)**	**% (SS/TS)**	**% (AS/SS)**
000	8158	7730	6087	1643	94,75%	78,75%
001	24118	9743	5807	3936	40,40%	59,60%
002	16727	305	214	91	1,82%	70,16%
003	15176	1085	983	102	7,15%	90,60%
004	25744	10645	6036	4609	41,35%	56,70%

Table 6
Telnet sessions per trial

	Telnet					
Trial ID	**Total Sessions (TS)**	**Successful Sessions (SS)**	**Active Sessions (AS)**	**Passive Session (PS)**	**% (SS/TS)**	**% (AS/SS)**
000	83118	82258	81679	579	98,97%	99,30%
001	152	0	0	0	0,00%	0,00%
002	459896	51008	11631	39377	11,09%	22,80%
003	190293	24418	7592	16826	12,83%	31,09%
004	138015	131514	131471	43	95,29%	99,97%

The data presented in Tables 5 and 6 show that attacker behavior differs depending on the protocol in use. Most observed attacks were conducted over the Telnet protocol. The main discrepancy between trial 001 and trial 004 relates to Telnet traffic, while no similarly significant differences are observed for SSH. Despite the two trials having identical configurations, trial 001 recorded no Telnet activity, whereas trial 004 detected approximately 140,000 Telnet sessions. This discrepancy may result from a configuration error or a software issue with the Telnet protocol module and requires further investigation.

By analyzing Telnet and SSH sessions separately, several trends emerge that should be further investigated in future tests. As for Telnet, excluding trial 001, a tighter set of constraints appears to result in a proportional increase in the total number of sessions. Conversely, for SSH, the highest traffic seems to occur when controls are limited to login credentials only. Trial 000 also shows promising results in terms of active sessions; however, it records a generally lower total number of sessions for both protocols compared to the other trials. This suggests that it primarily captures botnets operating with relatively unsophisticated agents that do not perform honeypot fingerprinting, and instead rely on broad, volume-based scanning rather than targeted intrusion attempts. This interpretation is further supported by the comparatively low absolute number of attacks observed in this trial. In contrast, trials implementing deception techniques and password-based access policy attract the highest volume of attacks. Stricter access policies are related to a higher volume of Telnet attack attempts. Excluding trial 001, the highest number of attacks was recorded in trials 002 and 003, followed by trials 004 and 000.

However, when examining the ratio of successful to active Telnet sessions, attackers appear to be more effective against systems with permissive access policies. In trials enforcing strict access control (trials 002 and 003), just over 10% of sessions gained access, with most remaining inactive. Conversely, in trials with permissive policies (trials 000 and 004), nearly all sessions successfully accessed the system and performed at least one action.

For SSH, data confirmed that the use of honeypot obfuscation techniques is associated with an increased number of detected intrusion attempts. Trial 000 registered the lowest level of traffic, followed by trials 004, 001, 002, and 003. When examining the ratio of successful to active SSH sessions, trials enforcing stricter access controls exhibit a lower proportion of successful login but a higher share of active sessions compared to trials with more permissive policies.

Overall, SSH tends to exhibit more selective behavior: fewer sessions gain access, but those that do are more likely to interact. To further investigate this hypothesis, we conducted a more in-depth analysis of SSH session results using the HASSH[10] library, which identifies and groups attacks based on the cryptographic fingerprint of the SSH protocol by generating a hash code. This allows correlation of distinct sessions from different IP addresses that, using the same SSH stack, may hypothetically belong to the same botnet. The hypothesis is that if a hash generates only active sessions or only passive sessions, we are dealing with botnets that either proceed with the attack or have detected the honeypot and take no further action.

Table 7 provides an overview of the findings. AS refers to active sessions, while PS indicates passive sessions.

Table 7

HASSH fingerprint analysis: correlation between SSH stack hashes and session behavior (successful vs. passive)

	Trial 000		Trial 001		Trial 002		Trial 003		Trial 004	
HASSH	**SS**	**PS**	**SS**	**PS**	**SS**	**PS**	**SS**	**PS**	**SS**	**PS**
01ca35584ad5a1b66cf6a9846b5b2821	3236	1								
03a80b21afa810682a776a7d42e5e6fb	2357	1501	7870	3832	92	4	218	8	9273	4460
04f9d9fca3a978a7c079aa3379fd21d6	2	1	2	1						
069fb483253fc12ecaf3731bac1ae020	1	1								
0a07365cc01fa9fc82608ba4019af499	850	11	74	5	78	2	200	8	412	6

[10] HASSH - https://github.com/salesforce/hassh

19532158b559096b89b1a5f7d17175b2	15	15	20	20	29	29	26	26	17	17
1bcedb0742d00117d5655f8a9729f9c7	1	1								
390ffe68a68c2a2891210413e80689fa	70	6					13	2	14	3
4e066189c3bbeec38c99b1855113733a	13	1	22	1						
98ddc5604ef6a1006a2b49a58759fbe6	7	1								
a37561952f836bcafd479cf44ccf6d52	9	9	10	10	5	5	4	4	11	11
acaa53e0a7d7ac7d1255103f37901306	310	4								
c118de82e19e5384f50f9bfd36c1a5dc	14	14	22	9	14	9	13	7	22	11
c1953cec4623dba5c2bdf4b16e831ea9	3	2								
e0a89321534ab8af0f512c4614c291e9	3	2	2	1						
e13ba222a14b8b916c6c32f9fd6aec5d	6	4	40	20					32	16
f0dae8afad40e79868006dda460a3625	150	41								
f555226df1963d1d3c09daf865abdc9a	27	18	9	4					75	37
fda360b1b4f4d3455cb75c6e7edb1d11	656	10	388	15	53	39	37	37	387	5
2ec37a7cc8daf20b10e1ad6221061ca5			881	1						
5f904648ee8964bef0e8834012e26003			18	2	30	2			20	1
6a77bbd6ef48d6a9959a47aa4a42a505			381	13			99	1	306	3
713bd9cc935561939c02dad25af2d3de			2	1			4	1	66	33
8c95e28f1643c38e5d64511b4d499e94			2	1						
1f2f2f9b0a736952f743e5490c64c98a					4	1				
1cc79c7da9b5d5eead2c60983332a556							2	1		
2f5ebb184f5d02324eed9a822f5259dc							2	2		
57e4cc8ee36c3d78f75c6a05acd55963							467	5		
1b8acd46a07d2dc9854db9ec4044c45c									6	4
3707f494e8d8ae4d370f0f7dc6d7d714									2	1
a2de0f306611e0957be704f5b0e35a82									2	1

From the analysis of Table 7, we can draw further observations that could be expanded through a more detailed analysis of the information associated with each identifier:

- Many passive sessions from SSH access are related to a single identifier (03a80b21afa810682a776a7d42e5e6fb), associated with a significant percentage of the password 3245gs5662d344, a default credential for the Polycom CX600 IP phone as noted in [14]. In trials 002 and 003, the same identifier shows almost no passive sessions, as this password is not among the first 100 or 500 default passwords respectively.
- Two identifiers, across all trials, generate exclusively passive sessions. We hypothesize these are targeted attacks that successfully detect the honeypot. Further analysis of the properties associated with these identifiers could be the subject of future research.
- Most identifiers are not detected across all trials, but only in some. This may be due to the limited test duration, attack patterns varying over time, or identifier changes caused by factors unrelated to honeypot operation.

- Some identifiers, in the trials where they are present, are characterized by very few passive sessions. In these cases, attackers failed to recognize the honeypot and proceeded with the attack.

5.2. Results Correlation

As further preliminary work, the creation of a CSV dataset was planned, using daily data extracted from Cowrie attack logs. The goal is to synthesize data for use as input to ML models. The final dataset is generated by aggregating and processing events from each session. In each CSV file, every row summarizes data from a single attack session, with the properties described in Table 8.

Table 8
CSV dataset properties: summary of features extracted for each attack session

Attribute	Explanation	Example
session	session identifier	b5958fcb723a
src_ip	attacker IP	178.16.54.6
username	login credential	admin
password	password credential	admin
protocol	SSH/Telnet	ssh
version	ssh version	SSH-2.0-libssh2_1.11.1
hassh	hash fingerprint of ssh client	56e3cc8ed46c3d68b76c6a05aad56964
duration	session duration	0,9
TM_DIF_MAX	maximum difference between event session timestamps (in seconds)	215
TM_DIF_MIN	minimum difference between event session timestamps (in seconds)	1
TM_DIF_MEA	difference mean of event session timestamps (in seconds)	0,07
TM_DIF_STD	standard deviation of event session timestamps	0,09
TM_DIF_MED	median of event session timestamps	58
history	summary of the session	SE.CO|CL.VE|CL.KE|LO.SU|SE.PA| CO.IN->uname -a|CO.IN->w|LO.CL|SE.CL
events	summary of event session	SE.CO|CL.VE|CL.KE|LO.SU|SE.PA|
default_cred	matching between attackers credential and default credential of main devices	[['Lucent']]

6. Conclusions

In this work, based on data collected across multiple experimental trials, we show that effective honeypot obfuscation enables the collection of a significantly larger volume of attack-related data compared to honeypots deployed with default configuration parameters. Moreover, obfuscation allows insight into botnet agents that actively perform honeypot fingerprinting. As a result, this approach leads to more representative collections of attack commands, within the inherent limitations of the platform, as Cowrie remains a medium-interaction honeypot and cannot fully emulate all commands available on a real operating system.

Additionally, the adoption of a suitably configured access control policy contributes to improved classification of the collected data. Although such policies restrict access from attacks using uncommon credentials, they help prevent honeypot identification by agents performing fingerprinting through random credential generation. At the same time, they reduce attacks specifically targeting narrow classes of devices that differ from the generic Linux-like environment partially emulated by Cowrie, as well as generic attacks relying on uncommon credentials.

A novelty of this work is the analysis of test sessions across five trial windows over a one-month observation period, which revealed that SSH and Telnet attacks differ substantially in both behavior and scale depending on the implemented access policy. For Telnet-based attacks, increasingly restrictive access constraints resulted in a higher proportion of passive sessions (i.e. sessions that did not generate any commands). Conversely, for SSH-based attacks, tightening credential restrictions led to a reduction in passive sessions. These findings indicate that the selection of credential types and the calibration of access constraints must be performed carefully, taking into account the intended analysis goals and the protocol under consideration. Notably, for the SSH protocol trial 003 emerged as the best trade-off between honeypot credibility and data quality, as the combination of obfuscation and a moderately restrictive access control policy resulted in a minimal number of passive sessions.

Another contribution is the correlation of sessions originating from distinct IP addresses that potentially belong to the same botnet through HASSH integration. This analysis revealed fingerprints associated exclusively with passive behavior and enabling improved filtering between device-specific and generic attacks. Finally, this work includes the development of a fully open-source framework for data parsing and visualization, extending log-based analysis with graphical components that support intuitive statistical insights.

7. Future Works

Future work will pursue several directions. First, an in-depth study of the collected data and correlations will be conducted, along with additional trials featuring mixed credential policies (combining default passwords with commonly used ones). Analysis of uploaded and downloaded malware will be performed, leveraging the VirusTotal[11] API for automated classification.

Further improvements to honeypot deception techniques will be explored, including the simulation of specific devices by accurately replicating their fingerprints (e.g., IP cameras, Raspberry Pi[12]). Commands executed by attackers during active sessions will be studied, potentially employing ML techniques for pattern recognition.

Finally, we aim to develop a classification system for botnets based on their ability to detect honeypots, assigning detection capability scores to different threat actors.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT 5.2 to: Grammar and spelling check. After using these services, the authors reviewed and edited the content as needed and took full responsibility for the publication's content.

[11] VirusTotal - https://www.virustotal.com
[12] Raspberry Pi - https://www.raspberrypi.com/

References

[1] T. Holz and F. Raynal, Detecting Honeypots and Other Suspicious Environments. In Proc. of the 6th Annual IEEE System, Man and Cybernetics Information Assurance Workshop SMC '05, 2005, retrieved from https://doi.org/10.1109/IAW.2005.1495930.

[2] A. Vetterl, R. Clayton, Bitter harvest: Systematically fingerprinting low and medium-interaction honeypots at internet scale. 12th USENIX Workshop on Offensive Technologies, WOOT 2018, Co-Located with USENIX Security 2018, retrieved from https://doi.org/10.17863/CAM.27923.

[3] M.R. Amal and P. Venkadesh, Review of Cyber Attack Detection: Honeypot System. Webology, Vol 19 no. 1, 2022, retrieved from https://doi.org/10.14704/web/v19i1/web19370.

[4] W. Ahmad, , M. A.Raza, S. Nawaz and F. Waqas, Detection and Analysis of Active Attacks using Honeypot. International Journal of Computer Applications, 184(50), 2023, retrieved from https://doi.org/10.5120/ijca2023922624.

[5] S. Srinivasa, J. M. Pedersen, E. Vasilomanolakis, Gotta Catch 'em All: A Multistage Framework for Honeypot Fingerprinting. Digital Threats, 4(3), Article 42 (2023), retrieved from https://doi.org/10.1145/3584976.

[6] P. Krajcik, M. Mikulas, P. Helebrandt and I. Kotuliak, Improvement of Cowrie honeypot interaction and deception capabilities. 2025 Communication and Information Technologies Conference Proceedings, KIT 2025 - 13th International Scientific Conference, retrieved from https://doi.org/10.1109/KIT67756.2025.11205433.

[7] M. Oosterhof, 2016, Cowrie SSH/Telnet Honeypot. Retrieved December 1, 2025 from https://github.com/cowrie/cowrie.

[8] T. Sethi and R. Mathew, A study on advancement in honeypot based network security model. Proceedings of the 3rd International Conference on Intelligent Communication Technologies and Virtual Mobile Networks, ICICV 2021, retrieved from https://doi.org/10.1109/ICICV50876.2021.9388412.

[9] M. B. Ozkok, B. Birinci, O. Cetin, B. Arief and J. Hernandez-Castro, Honeypot's Best Friend? Investigating ChatGPT's Ability to Evaluate Honeypot Logs. ACM International Conference Proceeding Series, 2024, retrieved from https://doi.org/10.1145/3655693.3655716.

[10] Z. Morić, V. Dakić and D. Regvart, Advancing Cybersecurity with Honeypots and Deception Strategies, Informatics, 12 (1), 2025, retrieved from https://doi.org/10.3390/informatics12010014.

[11] C. Hetzler, Z. Chen and T. M. Khan, Analysis of SSH Honeypot Effectiveness. Lecture Notes in Networks and Systems, 652 LNNS, 2023, retrieved from https://doi.org/10.1007/978-3-031-28073-3_51.

[12] W. Z. Cabral, C. Valli, L. F. Sikos and S. G. Wakeling, Advanced Cowrie Configuration to Increase Honeypot Deceptiveness. IFIP Advances in Information and Communication Technology, 625, 2021, retrieved from https://doi.org/10.1007/978-3-030-78120-0_21.

[13] F. Sadique, S. Sengupta, Analysis of Attacker Behavior in Compromised Hosts during Command and Control. IEEE International Conference on Communications, 2021, retrieved from https://doi.org/10.1109/ICC42927.2021.9500859.

[14] Z. Zhao, S. Srinivasa and E. Vasilomanolakis, SweetCam: An IP Camera Honeypot. CPSIoTSec 2023 - Proceedings of the 5th Workshop on CPS and IoT Security and Privacy, retrieved from https://doi.org/10.1145/3605758.3623495.

[15] A. Javadpour, F. Ja'fari, T. Taleb, M. Shojafar and C. Benzaïd, A comprehensive survey on cyber deception techniques to improve honeypot performance. In Computers and Security (Vol. 140), 2024, retrieved from https://doi.org/10.1016/j.cose.2024.103792.