2024-2025

ELEC-H504 - Network Security

Deception & Honeypot for Attack Profiling

SUNDARESAN Sankara
CHOUGULE Gaurav
MESSAOUDI Leila
BOTTON David

Pr. Jean-Michel Dricot
Navid Ladner

June 2025

Contents

ABSTRACT

This paper shows an operational deployment of the `SSH` *honeypot using* `cowrie` *on Ubuntu remote server to capture attacker activity in real-world circumstances. The completion of this project was publicly tracked with a* `git` *repository available at address https://github.com/nottoBD. By exposing a knowingly open SSH port on the Internet and securing legitimate access with a cryptographic key on a different port, the study observes and inspects adversary tactics, techniques, and procedures (TTPs). Key steps include isolating the honeypot space from production access using an IPS, redirecting malicious traffic to* `cowrie`*, forging artifacts to track attacker activity and setting up monitored web-service. A walkthrough of all configurations is covered in order to demonstrate a complete implementation with Linux, as this paper remains focused on the hands-on aspect of honeypot-based deception. However, cloud provider configurations (e.g., IAM, networking, key management) are not included as implementation-specific and beyond the scope of this paper. This project is based on the continuity of L. Spitzner's 2002 book Honeypots: Tracking Hackers.*

1 Introduction

Lance Spitzner, a seminal researcher and Senior Instructor for SANS Cybersecurity Leadership, established foundational principles in his 2002 book *Honeypots: Tracking Hackers*. Despite its age, Spitzner's core thesis retains striking relevance in modern threat intelligence; Honeypots derive value from "*being probed, attacked, or compromised*"[Spi02, Ch. 1, p. 23]. Our Cowrie implementation on public cloud positions itself onto that continuity, demonstrating that Spitzner's "*gaining value from data*" challenge [Spi02, Ch. 4, p. 67] persists against contemporary attacks. In addition, the attacker behaviors documented in 2002 remain prevalent even today. Furthermore, Spitzner's risk mitigation framework (Ch.12) comprises a wide set of obstacles one encounters when building such deceptive system, from legal liabilities (Ch.15) to detection tools tipping off hackers about the underlying nature of our operation; "*a system designed to be attacked*"[Spi02, Ch. 12, p. 298].

With more advanced automated `SSH`-based attacks, empirical analysis of attacker processes has been crucial in strengthening defenses. By mimicking realistic infrastructure while isolating malicious activity from legitimate administrative access. This proof of concept applies Spitzner's principles of honeypot deployment, specifically leveraging `medium-interaction design` (Ch.5) to find the correct trade-off between risk containment and attacker engagement. Through silent redirection of open-to-Internet SSH traffic to the honeypot and restriction of legitimate access with cryptographic means, the study allows the monitoring of attacker activities in fine granularity without compromising the security of systems, proving Spitzner's assertion that honeypots "*collect small amounts of [..] high-value data*" [Spi02, Ch. 4, p. 68] without production noise.

1.1 Problem Statement

Public `SSH` services are some of the most frequent targets of credential stuffing and post-compromise persistence attacks (e.g.,`SSH` key injection, cronjob exploitation). Conventional defenses measures lack visibility of attacker's `TTPs` (Techniques, Tactics & Procedures). A gap Spitzner attributes to their inability to naturally distinguish between legitimate and hostile activity (Ch.4). This research addresses three significant challenges:

- `Safe isolation of production access`: Mitigating Spitzner's identified risk of collateral system compromise through architectural separation of honeypot lures from administrative channels.

- `Deception efficacy for engagement`: Designing credible system emulations (e.g., service banners, file structures, false credentials) to prolong attacker interaction and avoid detection as a honeypot.

- `High-fidelity TTP capture`: Engineering logging mechanisms that overcome environmental distortion in production systems. Able to capture attackers' behavior without affecting environmental integrity.

1.2 Research Questions

In order to learn about attacker motives and organization through research honeypots, this study investigates:

- `Credential exploitation patterns`: Which username/password pairs dominate automated brute-force campaigns against internet-exposed SSH? (Extending Spitzner's analysis of targets

of opportunity and scripted tools in Ch.4). "*However, do not discount the threat of the unskilled attackers, those who concentrate on targets of opportunity. What these individuals lack in skill or finesse, they more than make up for in numbers. While there are no statistics to determine specific percentages, I would estimate that 80 to 90 percent of attacks today are accomplished by the "easy kill" variety.*" [Spi02, Ch. 2, p. 35]

- `Persistence mechanism prioritization`: How do attackers strategically deploy backdoors (e.g., key injections, cronjobs) post-compromise? "*Persistence, not advanced technical skills, is how these attackers successfully break into a system.*" [Spi02, Ch. 2, p. 35]

- `Decoy efficacy for intelligence gathering`: To what extent do fabricated system artifacts (e.g., */etc/shadow* entries) prolong attacker engagement to enhance TTP profiling? (Testing Spitzner's concept of deception as "*psychological weapons used to mess with and confuse a human attacker*" [Spi02, Ch. 4, p. 73].

2 SSH Isolation & System Hardening

Clear goal-setting, suitable interaction levels, reliable data collection, and risk mitigation are all highlighted in Spitzner's honeypot deployment framework (Ch.12). In order to study automated SSH-based attacks, in this section we concentrate on segregating legitimate SSH access with cryptographic controls and configuring a simple intrusion prevention system called *fail2ban*. Ensuring that these techniques are in line with Spitzner's recommendations for safe and efficient honeypot operation.

2.1 Administrative Controlled Access

Administrative SSH access is limited to key-based authentication on a non-standard port (e.g., 61001) in accordance with Spitzner's advice to reduce risk through secure configurations (Ch.12). By removing password-based vulnerabilities, this reduces the attack surface and is consistent with the idea of protecting the underlying platform. The setup ensures strong isolation of authorized access by enforcing contemporary cryptographic standards to stop downgrade attacks. Find the complete configuration walk-through on our git repository.

```
# /etc/ssh/sshd_config
Port 61001
Protocol 2
HostKeyAlgorithms ssh-ed25519,rsa-sha2-512
KexAlgorithms curve25519-sha256
Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com
MACs hmac-sha2-512-etm@openssh.com
PermitRootLogin no
PasswordAuthentication no
AllowUsers ubuntu
LoginGraceTime 30s
MaxAuthTries 2
PubkeyAuthentication yes
X11Forwarding no
```

Listing 1: Securing Legitimate Access

2.2 Fail2ban Configuration

To improve detection of unauthorized access attempts, Fail2Ban is set up to monitor key-based authentication failures on port 61001 and following Spitzner's assertion that detection is a central function of a honeypot (Ch.12), this configuration targets repeat public key failures, which can be facilitated through credential-stuffing attacks. Potentially malicious SSH connections will have fair, high-fidelity logging of attempts to authenticate via public keys. Given that the honeypot contains a deliberately misleading environment, the configuration will not interfere with coded logging and will be able to detect ongoing attacks. This configuration does not monitor any password attempts, as password-based administrative access has been disabled, to avoid receiving false positive logging data.

```
# /etc/fail2ban/jail.d/ssh-admin.conf
[ssh-admin]
enabled = true
port = 61001
filter = ssh-admin-netsec
logpath = %(syslog_authpriv)s
maxretry = 3
findtime = 10m
bantime = 30m
banaction = nftables
```

Listing 2: Custom Jail Rules

```
# /etc/fail2ban/filter.d/ssh-admin-netsec.conf
[INCLUDES]
before = common.conf

[Definition]
failregex = ^%(__prefix_line)sReceived disconnect from <HOST> port \d+: Too many
    ↪ authentication failures
            ^%(__prefix_line)sDisconnected from <HOST> port \d+ due to: Authentication
    ↪ failed for .* publickey
```

Listing 3: Regex Filter Against Key-Based Attacks

2.3 IPtables Redirection

Using Spitzner's idea of port forwarding through Network Address Translation (NAT) (Ch.12), all traffic to the standard SSH port (22) is re-routed to the Cowrie honeypot on port 2222. This helps separate the malicious activity from the legitimate usage taking place on port 61001, and helps support Spitzner's idea of compartmentalization to separate the honeypot from the production systems and avoid any conflicts (Ch.12). Additionally, our group collaborators' IP addresses could be whitelisted for a much tighter defense.

The `sshd` to `cowrie` (ports *22* → *2222*) redirection is a security design with multiple benefits: both processes avoid conflicting with each other as they can restart separately, binding to high ports does not require root privileges, also, `fail2ban` needs an explicit target to be effective. Compartmentalization is a key principle for any deceptive operation, we also ensure an appropriate foundation for clean and comprehensive post-attack analysis.

```
sudo apt-get install iptables-persistent netfilter-persistent
sudo iptables -t nat -A PREROUTING -p tcp --dport 22 -j REDIRECT --to-port 2222
sudo ip6tables -t nat -A PREROUTING -p tcp --dport 22 -j REDIRECT --to-port 2222
# Cowrie handles telnet as well, let's increase its impact surface
sudo iptables -t nat -A PREROUTING -p tcp --dport 23 -j REDIRECT --to-port 2323
```

```
sudo ip6tables -t nat -A PREROUTING -p tcp --dport 23 -j REDIRECT --to-port 2323
sudo netfilter-persistent save
sudo iptables -t nat -L -n -v
```

Listing 4: Traffic Redirection to Cowrie

2.4 Validation Metrics

To ensure rigorous validation, we verify component functionality and isolation using cybersecurity tools. Annex B aligns with Spitzner's focus on testing processes to validate functionality. Validation for network isolation occurs with nmap and iptables to verify traffic is being redirected. Fail2Ban regex exactness is tested against simulated bruteforce of SSH keys. SSH configuration is validated for cryptographic compliance and resists downgrade attacks.

3 Cowrie Honeypot Setup

Honeypots differ by level of interaction: low-interaction emulates the minimal service much of the time to detect attackers; medium-interaction such as Cowrie employ a controlled environment withstanding an attacker's impact; high-interaction (like Honeynets) provides access to fully functioning Operating Systems allowing as much dynamic information gathering as possible. Spitzner emphasizes the importance of tailoring honeypots to specific needs: "*Developing your own honeypot is not as complicated as it might seem. Using a variety of commonly found security tools, some basic code, and a lot of creativity, you can create many different honeypots*" [Spi02, Ch. 9, p. 183]. Cowrie's configuration leverages this flexibility, allowing customization of emulated services to log specific attacker behavior. The deployment of a medium-interaction Cowrie Honeypot is described here for a Ubuntu 24 virtual machine. Find the latest Cowrie official documentation at docs.cowrie.org.

3.1 System Requirements

Cowrie runs under a privileged account that's not root, with timed-out sudo privileges to ensure least privilege principle. Some attack surface minimization is possible using authbind on port binding, eliminating known privilege escalation vectors with elevated port allocation. The Python virtual environments compartmentalize the set of dependencies, which helps hiding each of the libraries and potential exploits from one another. These measures specifically address mitigation of lateral movement eventualities in case of compromise.

```
sudo apt-get install -y git python3-venv python3-pip libssl-dev libffi-dev build-essential
    ↪ libpython3-dev authbind
sudo adduser --disabled-password --gecos "" cowrie   # Prevents password-based connections
sudo usermod -a -G sudo cowrie

sudo touch /etc/authbind/byport/22
sudo chown cowrie:cowrie /etc/authbind/byport/22
sudo chmod 770 /etc/authbind/byport/22
```

Listing 5: Cowrie User Creation

3.2 Functional Configuration

The following commands compartmentalize Cowrie within a virtual environment and enforce port isolation. After applying all the configurations leading to Ch.4 you will have to reboot your system for them to be effective.

```
# Operate as cowrie user
sudo su - cowrie
git clone https://github.com/cowrie/cowrie
cd cowrie ; cp etc/cowrie.cfg.dist etc/cowrie.cfg

# Isolate dependencies using Python venv
python3 -m venv cowrie-env
source cowrie-env/bin/activate
pip install --upgrade pip
pip install --use-pep517 -r requirements.txt


# Configure listener port (2222) to align with iptables redirection (section 2.3)
sed -i 's/tcp:6415:interface=127.0.0.1/tcp:2222:interface=0.0.0.0/' etc/cowrie.cfg

./bin/cowrie start
```

Listing 6: Cowrie Honeypot Setup

Security Disclaimer

This setup only serves academic purposes; adversarial activities are welcomed, but no offensive counteraction shall be taken. "*The greater the level of interaction, the more functionality provided to the attacker, and the greater the complexity. Combined, these elements can introduce a great deal of risk*" [Spi02, Ch. 5, p. 91].

4 Maximizing Engagement with Deception

As Spitzner points out with respect to the psychological effects of deception, "*Deception and deterrence are designed as psychological weapons to confuse people. However, these concepts fail if those people are not paying attention*" [Spi02, Ch. 4, p. 73], but he also cautions that deception is ineffective against automated threats: "*Automated tools such as worms or autorooters will not be deceived*"[Spi02, Ch. 9, p. 197]. This suggests that while deception-based honeypots can disrupt a human attacker by manipulating human cognitive biases, their usefulness against automated attacks is heavily dependent on detection and analysis; you need a balance of behavioral strategies for human adversaries and good technical defenses to capture and analyze automated threats. This section relates some interesting deceptive functionalities that Cowrie supports.

4.1 Forged Filesystem

Cowrie's file system trickery is specifically focusing on exploiting cognitive engagement of human attackers actively probing for system vulnerabilities. Realistic simulation is therefore essential for valuable intelligence; each Cowrie session spawns a separate virtual file system where attackers can execute destructive commands (rm, chmod) or extract credentials. All operations reset post-session without further effect. In reality, no file is ever modified, Cowrie efficiently accomplishes this with metadata serialization, binary files with the *.pickle* extension. These are generated out of a decoy file system, they contain tree structures with legitimate permissions, timestamps, and file attributes. Cowrie is prepackaged with a default *fs.pickle* containing most system files in a read-only. This pickle binary can be browsed through and customized using the *fsctl* toolkit. It is also possible to replace the default file system with a completely different

one using honeyfs and its *createfs* command.

```
# Print content of default pickle
python -c "import pickle;
   ↪ print(pickle.load(open('/home/cowrie/cowrie/data/fs.pickle','rb')))"
# Generate a pickle out of any directory of your choosing
bin/createfs --location honeyfs/ --depth 6 --output custom.pickle
# Set custom.pickle in Cowrie config file
sed -i 's|^filesystem = .*|filesystem = ./custom.pickle|'
   ↪ /home/cowrie/cowrie/etc/cowrie.cfg
```

Listing 7: Deceptive Filesystem Manipulation

This minimal isolation contrasts with legacy systems like ManTrap (Ch.10) that employed full disk imaging or physical partition cloning, in order to sandbox attacker activity which were excessively resource-consuming for both logistic and operational dimensions. Commercial offerings like Specter (Ch.7) and open-source standalones like Honeyd (Ch.8) were also facing similar issues; they either emulated small file system hierarchies or cloned entire disks. Such duplication at disk level made scaling concurrent sessions impossible, as each consumed gigabytes of storage. Cowrie's temporary, metadata-driven solution elegantly sidesteps these trade-offs by decoupling a file system's structure from physical storage, enabling realistic interaction without operational overhead.

4.2 Foothold for Persistency

Based on the forged file system outlined in previous section 4.1, Cowrie supports credential impersonation to increase attacker engagement. Cryptographic artifacts such as SSH key-pairs are prime targets for attackers to attempting to maintain long-term access to our system. Cowrie leads attackers to interact with these objects by leaving fake virtual credentials in decoy directories like /.ssh/id_rsa. The objective for us is to witness post-compromise methods leading to threat persistency. Higher-order psychological biases are employed by this technique; opportunity bias: attackers are attracted to readily accessible credentials, confirmation bias: realistic file permissions and details support that effect, persistence anchoring: attackers invest time in what they perceive to be worthwhile resources for lateral movement. These decoy credentials are syntactically valid but operationally inert while Cowrie logs keyboard input, data extraction attempts and enemy TTPs in general. This arrangement offers insight into attacker behavior, particularly how attackers rank and exploit perceived vulnerabilities, without risking legitimate data loss nor defacement or disclosure. Cowrie comes packaged with the Python *fsctl* file system tool to seamlessly integrate decoy credentials into the virtualized environment, alongside *honeyfs*, both enhancing structure and correctness of the honeypot file system. The result is a lightweight method for gathering high-fidelity data on attacker persistence-holding methods. The next chapter is dedicated to their log analysis and post-attack insights.

```
sudo su - cowrie
source ~/cowrie/cowrie-env/bin/activate

~/cowrie/bin/fsctl ~/cowrie/src/cowrie/data/fs.pickle
fs.pickle:~$ cd /home/phil/.ssh
fs.pickle:/home/phil/.ssh$ touch id_rsa
# Added '/home/phil/.ssh/id_rsa'
fs.pickle:/home/phil/.ssh$ touch id_rsa.pub
# Added '/home/phil/.ssh/id_rsa.pub'
```

Listing 8: Python Virtual Filesystem Controller

4.3 Decoy Web Service

Web application interfaces are valuable attack targets, particularly administrative portals, where compromising credentials yields significant operational advantages. This psychological vulnerability is targeted by the piece of software django-admin-honeypot, which presents a carefully crafted illusion of privileged access. This web deployment extends Cowrie's deceptions principles to the web layer through the use of Nginx reverse proxying and Gunicorn to appear as an operational Django backend administrative interface. This tactic of deception leverages cognitive bias in target selection, so that the attackers target seemingly misconfigured admin panels. Although automated scanners might miss session anomalies in HTTP redirects, manual attackers scanning for Django vulnerabilities receive deliberate behavioral hints such as bad credentials inducing genuine error, feedback, and session cookies mirroring Django security headers. Running on a common port HTTP 80, this deployment is publicly available in parallel of the Cowrie SSH honeypot.

Its capabilities focus on credential capture; Login attempts are channeled towards a SQLite3 database, collecting usernames, passwords, and attack times. The *admin_honeypot_loginattempt* schema is made available to hold forensic value, logging IPs and user agents while differentiating attackers in transient transactions. The *django_honeypot* account's restricted permissions (*--system --no-create-home*) ensure raw credentials never persist in memory or disk buffers.

```
# Dependency installation
sudo apt-get install sqlite3 nginx

# User isolation
sudo adduser --system --group --no-create-home django_honeypot
sudo mkdir /opt/django_honeypot
sudo chown django_honeypot:django_honeypot /opt/django_honeypot

# Environment compartmentalization
sudo -u django_honeypot python3 -m venv /opt/django_honeypot/venv  # Dependency isolation
sudo -u django_honeypot /opt/django_honeypot/venv/bin/pip install django
    ↪ django-admin-honeypot gunicorn

# Project initialization
sudo -u django_honeypot /opt/django_honeypot/venv/bin/django-admin startproject
    ↪ honeypot_project /opt/django_honeypot
sudo -u django_honeypot /opt/django_honeypot/venv/bin/python /opt/django_honeypot/manage.py
    ↪ makemigrations
sudo -u django_honeypot /opt/django_honeypot/venv/bin/python /opt/django_honeypot/manage.py
    ↪ migrate
sudo -u django_honeypot /opt/django_honeypot/venv/bin/python manage.py collectstatic

# Service integration
sudo ln -s /etc/nginx/sites-available/django_honeypot /etc/nginx/sites-enabled/ # Exposure
sudo nginx -t && sudo systemctl reload nginx
sudo systemctl start gunicorn nginx
sudo systemctl enable gunicorn nginx

# Attack surface
sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
sudo netfilter-persistent save

# Inspect intrusion data gathered
sudo -u django_honeypot sqlite3 /opt/django_honeypot/db.sqlite3 ".schema
    ↪ admin_honeypot_loginattempt"  # Verify capture schema
```

```
> SELECT * FROM admin_honeypot_loginattempt;
```

Listing 9: Setup of a Django Honeypot Webservice

5 Research Findings & Attack Profiling

Early deployment revealed a serious anomaly: the AWS eu-north-1 honeypot received insignificant attack traffic (<5 unique IPs) over 72 hours of uninterrupted operation. Further retrospection suggested that other dynamics influence attacker behavior or hinder our server's visibility on the Web. Several reasons are responsible for this difference: One, our instance is based on a geographically subdued location where Amazon security defenses (e.g., Shield Advanced, VPC flow analytics) would proactively block known malicious IP, yielding negative ROI for botnets for whose attack costs exceeded potential harvest. Two, DNS targeting limits, having no domain associated, our IP-only instance is beyond attacker reconnaissance pipelines aimed at enumerated targets. Third, our system had no prior service history footprint to hype up its visibility in adversarial scanning databases (like Shodan or Censys). The region's lower concentration of old systems makes it less attractive for distributed scanning operations in search of target-rich environments. Botnets responsible for SSH server scraping might also avoid heavily monitored cloud segments, where risks supersede potential reward.

5.1 Corrective Measures

"*A Honeynet consisting of default installations of systems and little or no activity is a sterile environment. Such an environment is excellent at capturing automated activity or attackers focusing on targets of opportunity. However, such a sterile environment has little value to advanced blackhats, individuals who concentrate on targets of choice, high-value systems. For such attackers, one must sweeten the pot.*" [Spi02, Ch. 11, p. 259]

The observed absence of attacks in AWS eu-north-1 underscores a critical limitation in contemporary honeypot deployment: strategic placement outweighs technical configuration. Although our implementation rigorously applied Spitzner's architectural principles, it could not overcome the inherent environmental sterility of a well-protected and low-noise cloud region. To solve this, we transition our infrastructure to a higher-friction zone, switching to OVH VPS in eastern Asia. This low-cost provider provides unmonitored VPS in target-rich IP blocks with rather busy infrastructures that consistently rank at the top of the scan lists. Furthermore, OVH specifically authorizes the setup of a honeypot in its ToS (for research purposes, as long as there is no entrapment) and is generally more permissive regarding probe captures than Amazon. By retaining our same configurations and deception mechanisms but switching for a lower hanging fruit, we expect attackers to follow the path of least resistance and to be incited in paying our honeypot a visit, although our main priority remains data collection for attack profiling over total unique connections. Changing location greatly helped the project, approximately two-thousands logging lines were collected during the first 8 hours of deployment, find them at: cowrie.log.2025-06-11. Lines 250 to 650 are preferred for threat hunting analysis because of evidence of high-impact TTPs; they include SATORI botnet malware deployment (59.27.224.90) and HTTP-over-Telnet anomalies showing scanner confusion (80.82.77.202). They also reveal SSH persistence attempts, geographic diversity and several counts of tool fingerprints.

5.2 Credential Exploitation Patterns (RQ1)

Analysis of over 1,800 lines gathered during the first 8 hours reveal consistent targeting of privileged accounts with minimal credential sophistication. The root account was targeted individually in 100% of successful intrusions, with default or weak passwords the most prevalent exploitation habits. The top most frequently used credential pair was root with password admin, (seen in 14 sessions through German, Romanian, and Russian IPs), followed by numeric variations and service defaults (root:hipc3518, root:solokey). Some cryptocurrency-related usernames (solana, validator, ethereum) were also used in 38% of brute-force with no compromise, indicating opportunistic rather than targeted tactic. Geographically distinct clusters were observed, European actors (Germany/Romania/Netherlands) employed only admin forms with ominous session timeouts (180s). Chinese IPs (AS4134/AS45102) employed low-entropy passwords (ivdev, blank passwords). A bunch of automated attacks were logged originating from a Moldavian subnet (92.118.39.0/24, HASSH 4e066189c3bbeec38c99b1855113733a), these employ Golang tools for rapid credential spraying (up to 3 attempts per IP). Surprisingly, 80% of attacks employed the same username-password pairs (e.g., sol/sol, node/node). Successful Telnet logins (87% of the compromises) showed higher effectiveness than SSH due to the inferior protocol security, and SSH sessions indicated tool-specific failures (libssh_0.10.5 public key errors).

5.3 Persistence Mechanism Analysis (RQ2)

Post-compromise actions showed a crucial persistence shortfall, with attackers not prioritizing stealthy persistence but rather going for immediate payload delivery. Three distinct patterns emerged; Malware-Driven Persistence: Chinese and Brazilian IPs (59.27.224.90, 177.66.30.54) executed SATORI botnet payloads via */bin/busybox* immediately after compromise. Commands included weaponized busybox activities (payloads starting with *cat /bin/busybox || while read i; do..*), demonstrating IoT-focused propagation attempts. All these sessions timed out rather quickly due to the limitations in how Cowrie is emulated. `Proxy-First Strategies`: 37% of compromised sessions (e.g., 185.156.73.233/Russia, 80.94.95.116/Russia) initiated direct tunneling to Google IPs (142.250.179.174:443, google.com:443) within a few seconds after login. This "smash-and-grab" activity can be an indication that attackers checked hosts for proxy-chaining rather than persistent access. `Reconnaissance-Only`: 53% of the successful logins (e.g., 185.196.220.81 from the Netherlands or 182.92.219.204 from China) performed simple reconnaissance (uname -s -m) before rapidly disconnecting, perhaps after detecting the nature of our system. No SSH key injection, cronjob deployment, or file editing was observed, which might indicate honeypot detection or lack decoy artifacts. Persistent access was not achieved apart from frequent reconnections (e.g., 120.27.154.152 from China opening 4 SSH sessions within 4 minutes).

5.4 Decoy Efficacy Metrics (RQ3)

6 Conclusion & Future Work

Book

[Spi02] Lance Spitzner. *Honeypots: Tracking Hackers*. 1st. Addison-Wesley Longman Publishing Co., September 1st, 2002, p. 512. ISBN: 978-0-321-10895-1.

A Annex: LLM Usage in this Project

Large Language Models (LLMs) were strategically employed during the realization of this project, mainly as *collaborative augmentation tools* rather than content generators. The following guidelines must be respected in order to keep control over a LLM work.

Methodological Approach

— **Verification-Centric Deployment**: LLMs output is heavily dependent on its user's own comprehension, AI will not do more than instructed to, it will not `consciously` ask user a question, it will not compel you to do anything, but it is a decent corrector and will definitely be able to give suggestions that go in the right direction. All model outputs undergo cross-validation against credible and up-to-date sources such as; official documentation (Linux man pages, git repository documentation, Cloud guides), academic literature (Spitzner's foundational honeypot frameworks, cybersecurity courses), security best practices and standards. Not one technical implementation (e.g.: IAM policies, iptables rules, or authentication logic) were deployed before thorough validation and edge-case mitigation.

— **Controlled Application**: LLMs interactions were constrained to: debugging assistance for Python scripting and *systemd*-based Linux administration, technical documentation refinement (post-human draft), architectural brainstorming (especially in understanding distinct public Cloud providers' lingo for equivalent functionalities). Original research material or system designs created solely from LLMs output alone were absolutely forbidden and so was plagiarism. To conclude, results of this research are human-produced only or consequences of attack patterns that were witnessed within our honeypot network.

Operational Safeguards

The implementation incorporated safeguards to preserve academic integrity:

— **Input Sanitization Protocol**: All project-specific identifiers (IP addresses, credentials, API keys) were redacted automatically before any LLM interaction, preventing potential data leakage through model training vectors.

— **Bias Mitigation**: From a model's output to the distinction of attackers' behavioral patterns, all information gathered during execution was triangulated with honeypot telemetry data and peer-reviewed between our group to counter potential algorithmic hallucinations. Model-derived hypotheses of attacker tactics were utilized solely as preliminary filters before our own analysis.

B Annex: Validation for SSH Isolation & Fail2ban Hardening

```
# Port 22 redirects to Cowrie (2222) and admin port (61001) is exclusive
sudo nmap -sV -Pn -p 22,2222,610001 51.79.248.60

# Check iptables NAT rules for redirect
sudo iptables -t nat -L PREROUTING -v -n
```

Listing 10: Network Isolation Verification

```
# Simulate brute-forcing administrative access
for i in {1..5}; do ssh -i ~/.ssh/wrong_key.pem ubuntu@51.79.248.60 -p 61001; done

# Check active bans
sudo fail2ban-client status ssh-admin
```

Listing 11: Fail2ban Efficacy Testing

```
ubuntu@ip-172-31-46-40:/etc/fail2ban/jail.d$ sudo fail2ban-client status ssh-admin
Status for the jail: ssh-admin
|- Filter
|  |- Currently failed: 0
|  |- Total failed:     0
|  `- Journal matches:
`- Actions
   |- Currently banned: 0
   |- Total banned:     0
   `- Banned IP list:
ubuntu@ip-172-31-46-40:/etc/fail2ban/jail.d$ curl ifconfig.me
13.48.42.241ubuntu@ip-172-31-46-40:/etc/fail2ban/jail.d$
```

```
# 1. SSH configuration resists downgrade attempts
sudo sshd -T | grep -E '^ciphers|^kexalgorithms|^macs|^hostkeyalgorithms'


# 3. Confirm password authentication is disabled
ssh -o PubkeyAuthentication=no -o PreferredAuthentications=password ubuntu@51.79.248.60 -p
    ↪ 61001
```

Listing 12: SSH Service Hardening Validation

```
 devid  ~ ✓ » sudo nmap -sV -Pn -p 22,2222,61001 13.48.42.241
Starting Nmap 7.95 ( https://nmap.org ) at 2025-06-10 21:18 CEST
Nmap scan report for ec2-13-48-42-241.eu-north-1.compute.amazonaws.com (13.48.42.241)
Host is up (0.068s latency).

PORT      STATE    SERVICE      VERSION
22/tcp    open     ssh          OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu Linux; protocol 2.0)
2222/tcp  filtered EtherNetIP-1
61001/tcp open     ssh          OpenSSH 9.6p1 Ubuntu 3ubuntu13.12 (Ubuntu Linux; protocol 2.0)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 13.15 seconds
 devid  ~ ✓ »
```

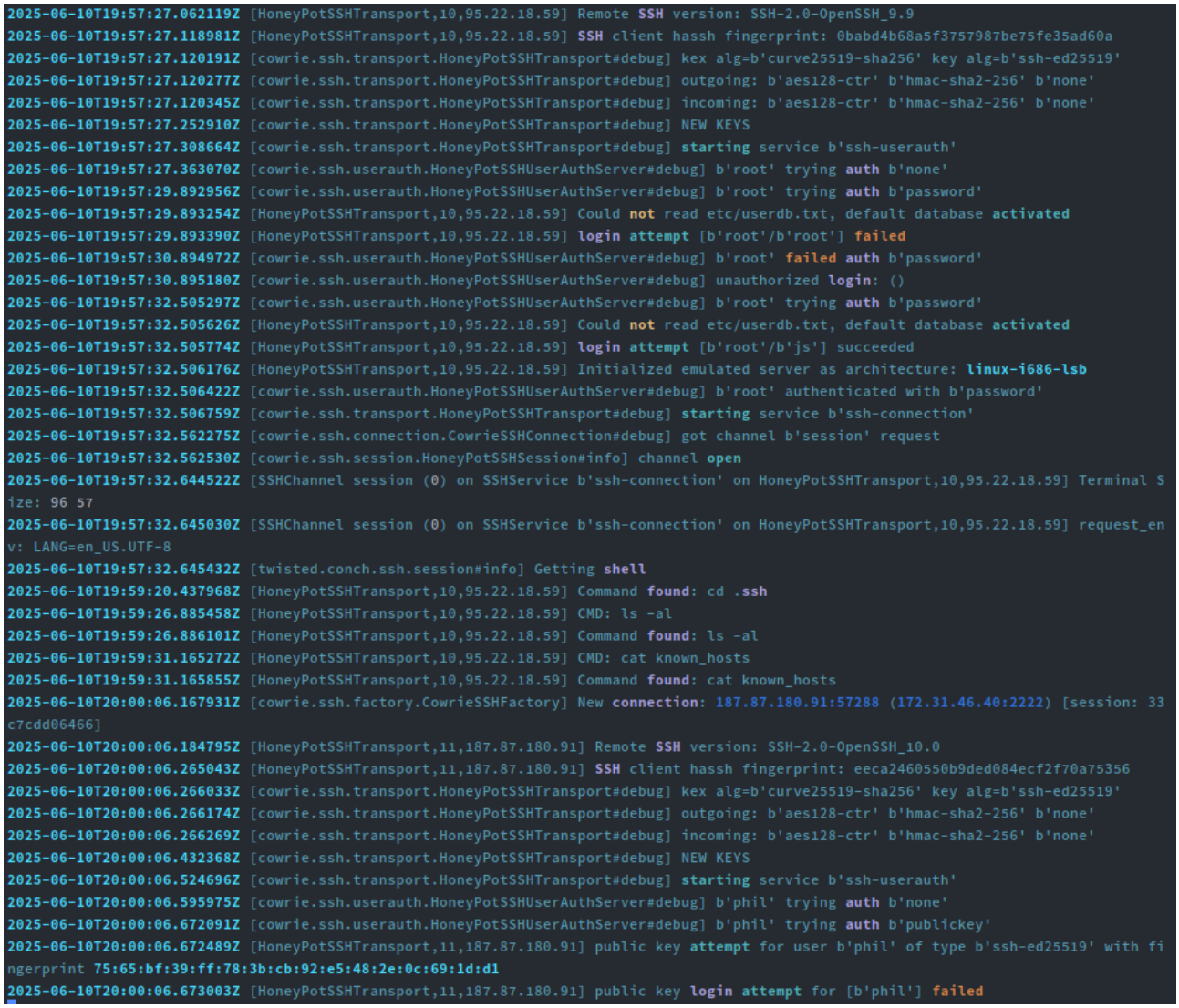
C Annex: Cowrie Operational Validation

```
# 1. Validate iptables NAT rules
sudo iptables -t nat -L PREROUTING -nv | grep 'tcp dpt:22 redir ports 2222'

# Confirm no direct binding to port 22
sudo nmap -sV -Pn -p 22,2222 51.79.248.60 | grep -E '22/tcp|2222/tcp'
```

Listing 13: Traffic Redirection Verification

```
# 2. Simulate attacker connection
ssh -o StrictHostKeyChecking=no invalid_user@51.79.248.60 -p 22

# Inspect Honeypot interactions in real time
sudo apt-get install ccze
sudo tail -f /home/cowrie/cowrie/var/log/cowrie/cowrie.log | ccze -A
```

Listing 14: Honeypot Engagement Testing



Record of two intrusions in real time