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 an Ubuntu public Cloud instance to capture attacker activity in real-world circumstances. 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 a malicious 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. Our public `git` repository is available at that location.

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" (p. 23). Our Cowrie implementation on public cloud positions itself onto that continuity, demonstrating that Spitzner's "gaining value from data" challenge (Ch.4) 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, such as legal liability (Ch.15) or signature-based detection tools tipping off hackers about the underlying nature of our operation; "a system designed to be attacked" (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" (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" (Ch.3, 12).

- `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).

- `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." (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" in Ch.4, p. 74).

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.: 2223) 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 2223
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 2223 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     = 2223
 filter   = sshd
 maxretry = 3
 findtime = 10m
 bantime  = 30m
```

Listing 2: Custom Jail Rules

```
 # /etc/fail2ban/filter.d/sshd.conf
failregex = %(cmnfailre)s
            <mdre-<mode>>
            ^%(__prefix_line)s(?: Received disconnect from <HOST> port \d+: Too many
    ↪ authentication failures | Disconnected from <HOST> port \d+ due to: Authentication
    ↪ failed for .* publickey )
            %(cfooterre)s
```

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 2223, 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 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
 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" (Ch.9). 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.

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 mitigating 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. Find the latest Cowrie official documentation at docs.cowrie.org.

```
# Operate as cowrie user
sudo su - cowrie
git clone https://github.com/cowrie/cowrie
cd cowrie

# Isolate dependencies using Python venv
python3 -m venv cowrie-env
source cowrie-env/bin/activate
pip install --upgrade pip
pip install --upgrade -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
```

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" (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" (Ch.4, p. 73), but he also cautions that deception is ineffective against automated threats: "Automated tools such as worms or auto-rooters will not be deceived" (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 Cowrie supports.

4.1 Forged Filesystem

Cowrie's filesystem 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 filesystem where attackers can execute destructive commands (rm, chmod) or exfiltrate credentials. All operations reset post-session without further effect. It efficiently accomplishes this with metadata serialization, binary files with the *.pickle* extension. These are generated out of a decoy filesystem, they contain tree structures with legitimate permissions, timestamps, and file attributes. Cowrie is prepackaged with a default *fs.pickle* that is well rounded already. Using honeyfs and its *createfs* command, we are able to integrate the actual content files for the decoy filesystem.

```
# 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 -l honeyfs/ -d 4 -o 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 standalone like Honeyd (Ch.8) were also facing similar issues; they either emulated small filesystem 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 filesystem's structure from physical storage, enabling realistic interaction without operational overhead.

4.2 Malicious Web Service

Web application interfaces are valuable attack targets, particularly administrative portals where credential compromise 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. While automated scanners might miss the 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, user agents, and brute-force modes 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. This implementation practices ethical honeypoting by avoiding credential reuse.

```
# Dependency Installation
sudo apt-get install sqlite3

# Secure 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

# 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
sudo systemctl enable gunicorn

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

# View Attack Data
sudo -u django_honeypot sqlite3 /opt/django_honeypot/db.sqlite3 ".schema
    ↪ admin_honeypot_loginattempt"  # Verify capture schema
    > SELECT * FROM admin_honeypot_loginattempt;
```

Listing 8: Ubuntu Setup of a Django Honeypot Webservice

4.3 Fabricated Cryptographic Credentials

5 Analysis: Attack Pattern & TTPs

5.1 Credential Exploitation Patterns

5.2 Persistence Mechanism

5.3 Decoy Efficacy Validation

6 Conclusion & Future Work

A Annex: LLM Usage for this Project

Methodological Approach

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 are to be respected in order to keep control over LLMs for work.

— **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, AWS Guides), academic literature (Spitzner's foundational honeypot frameworks, MS 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. Finally, results of the research are human-produced only and the result of attack patterns that were indeed 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**: Model output to attacker behavior patterns was triangulated with honeypot telemetry data and peer-reviewed threat intelligence reports to counter potential algorithmic hallucinations. Model-derived hypotheses of attacker tactics (e.g., SSH bruteforce patterns were utilized solely as preliminary filters on our empirical honeypot data analysis.

B Annex: Validation for SSH Isolation & Fail2ban Hardening

```
# Verify port 22 redirects to Cowrie (2222) and admin port (2223) is exclusive
sudo nmap -sV -Pn -p 22,2222,2223 $EC2_PUBLIC_IP

# Check iptables NAT rules for redirect (should show 22 to 2222)
sudo iptables -t nat -L PREROUTING -v -n

# Ensure no SSH service binds to port 22 (only Cowrie on 2222)
sudo ss -tulpn | grep -E ':22|:2222|:2223'
```

Listing 9: Network Isolation Verification

```
# Simulate key-based brute-forcing to trigger fail2ban
for i in {1..5}; do ssh -i ~/.ssh/wrong_key.pem ubuntu@localhost -p 2223; done

# Verify fail2ban logged the bans (look for 'ssh-admin' jail)
sudo grep "ssh-admin" /var/log/fail2ban.log

# Check active bans (should list test IP)
sudo fail2ban-client status ssh-admin
```

Listing 10: Fail2ban Efficacy Testing

```
# 1. Verify active SSH configuration matches hardening intent (no fallback to weak
    ↪ protocols)
sudo sshd -T | grep -E '^ciphers|^kexalgorithms|^macs|^hostkeyalgorithms'

# 2. Test SSH service for protocol/cipher negotiation weaknesses
nmap -Pn -p 2223 --script ssh2-enum-algos $EC2_PUBLIC_IP | grep -A 10 "algorithm
    ↪ negotiation"

# 3. Confirm password authentication is globally disabled (even if bypass attempted)
ssh -o PubkeyAuthentication=no -o PreferredAuthentications=password ubuntu@$EC2_PUBLIC_IP
    ↪ -p 2223
```

Listing 11: SSH Service Hardening Validation

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 $EC2_IP | grep -E '22/tcp|2222/tcp'
```

Listing 12: Traffic Redirection Verification

```
# 2. Simulate attacker connection
ssh -o StrictHostKeyChecking=no invalid_user@$EC2_IP -p 22

# Verify session capture in logs
sudo journalctl -u cowrie -f | grep 'SSH connection closed'
```

Listing 13: Honeypot Engagement Testing

```
# 3. Confirm execution context
ps -ef | grep cowrie | grep -v grep | awk '{print $1}' | uniq
```

Listing 14: Process Isolation Validation