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* `EC2` *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* `fail2ban`*, redirecting malicious traffic to* `cowrie` *via* `iptables`*, and forging artifacts to track attacker activity. A walkthrough of the settings is covered to demonstrate a complete implementation in* `cowrie`*, as this paper remains focused on the hands-on aspect of honeypot-based deception. This project's public* `git` *repository is available at that location.*

1 Introduction

As defined by spitzner2002honeypots, "a honeypot is a security resource whose value lies in being probed, attacked, or compromised" (p. 58).

Lance Spitzner, a 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 AWS positions itself onto that continuity, demonstrating that Spitzner's "gaining value from data" challenge (Ch.4) persists against contemporary attacks. Furthermore, Spitzner's risk mitigation framework (Ch.12) comprises the obstacle of signature-based detection tools that inform attackers about the nature of our operation, "a system designed to be attacked" (p. 298). Also, attacker behaviors documented in 2002 remain prevalent even today. 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-the-public SSH traffic to the honeypot and restriction of legitimate access with key-based authentication, the study allows the monitoring of attacker activities in fine granularity without compromising the security of systems, proving Spitzner's assertion that honeypots provide *«small amounts of high-value data»* 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). Existing defensive measures lack visibility of attacker's `TTPs` (Techniques, Tactics & Procedures), a gap to deeply analyze these malicious processes. This research addresses two significant challenges: (1) safe isolation of production access from honeypot bait to prevent collateral system compromise, and (2) the engineering of logging mechanisms that are able to capture attacker TTPs without affecting environmental integrity.

1.2 Research Questions

The report investigates three core questions: First, which credential pairs are most frequently exploited in SSH brute-force campaigns against internet-exposed systems? Second, what persistence mechanisms (e.g., unauthorized key additions, scheduled task manipulation) do attackers prioritize following initial compromise? Third, to what extent can strategically placed decoy files, such as fabricated /etc/shadow entries, extend attacker engagement to improve TTP profiling?

Security Disclaimer

This setup only serves academic purposes; adversarial activities are welcomed, but no offensive counteraction shall be taken in return. Additionally, cloud provider configurations (e.g., IAM, networking) are not included as implementation-specific and beyond the scope of this paper.

2 SSH Isolation & Fail2ban Hardening

2.1 Admin SSH Hardening

Restricts access to key-based auth on non-standard *port 2223* for legitimate contributors to the project, thus reducing attack surface.

```
# /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

Targets repeated public key failures, common in credential stuffing. Ignores password attempts (disabled in SSH).

```
# /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

Redirects all *port 22*'s traffic to `cowrie`, with rules persisting after reboot. Legitimate key-based accesses on *port 2223* are separated and preserved, we could additionally whitelist our group collaborators' IP addresses for a much tighter defense but will not do it here for ease of development. The `sshd` to `cowrie` (ports *22* → *2222*) redirection is a security design with multiple benefits: we avoid both processes to conflict with each other, we can restart them 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. Find IPtables' official documentation here.

```
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 apt install iptables-persistent netfilter-persistent
sudo netfilter-persistent save
```

Listing 4: Traffic Redirection to Cowrie

2.4 Validation Metrics

To ensure rigorous validation, we verify component functionality and isolation using cybersecurity tools. The following Annex provides exact commands for: Network isolation (nmap, `iptables`), `fail2ban` regex precision (ssh-key brute-forcing), `SSH` configuration enforcement and resilience regarding downgrade attacks, protocol validation and cryptographic compliance.

TODO: Validate that these commands work once cowrie is running on 2222

3 Cowrie Honeypot Deployment

3.1 System Preparation

A specific non-root user and a Python virtual environment isolate Cowrie's operations and therefore ensure that a least-privilege account reduces lateral movement threats in case of compromise. Find the latest official documentation at Cowrie Documentation

```
sudo apt-get install -y git python3-venv python3-pip libssl-dev libffi-dev build-essential
    ↪ libpython3-dev authbind # to bind privileged ports without root priv
sudo adduser --disabled-password --gecos "" cowrie    # prevents password-based connections
```

Listing 5: Cowrie User Creation

3.2 Honeypot Configuration

Compartmentalize Cowrie within a virtual environment and enforce port isolation

```
# 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 -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

3.3 Security Considerations

The rollout enforcing multiple layers of protection: Cowrie executes under a non-root privileged account with timed-out sudo privileges, implementing the least privilege principle. Attack surface minimization is offered by authbind on port binding, eliminating traditional root escalation threats by virtue of privileged port allocation. Compartmentalization is used for dependency isolation through Python virtual environments, preventing library conflicts and encapsulating potential exploits. Finally, fail2ban integration (Section **??**) protects administratively selectively protects port 2223 with ongoing uncontrolled attacker interaction with the honeypot at port 2222.

A 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 7: Network Isolation Verification

TODO: screenshot here

```
# Simulate key-based brute-forcing to trigger fail2ban
for i in {1..5}; do ssh -i /path/fake_key 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 8: Fail2ban Efficacy Testing

TODO: screenshot here

```
# 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 9: SSH Service Hardening Validation

TODO: screenshot here

B 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 10: 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 11: Honeypot Engagement Testing

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

Listing 12: Process Isolation Validation

C Annex: LLM Usage in this Project

Large language models (LLMs) provided targeted support during this honeypot deployment, strictly limited to non-operational tasks. For documentation, LLMs assisted in drafting initial LaTeX templates for technical sections such as SSH hardening (2.1) and iptables redirection (2.3), with all configurations manually validated against AWS and Cowrie documentation. During troubleshooting, models proposed diagnostic commands for SSH permission conflicts (e.g., 'chmod' adjustments in §3.2), which were later tested in isolated Docker environments before EC2 implementation. LLMs were explicitly excluded from security-critical decisions: firewall rules, Fail2ban thresholds, and cryptographic settings in '/etc/ssh/sshd_config' derived exclusively from NIST guidelines and Mozilla Infosec recommendations. No model influenced attacker engagement strategies, log analysis of captured TTPs, or live system interactions. All AI-generated content underwent peer review by the project's cybersecurity team, with particular scrutiny applied to network redirection mechanics and user permission workflows. Final configurations reflect human expertise, with LLMs serving solely as productivity accelerators for non-sensitive administrative tasks.