Catch-and-Log: A Lightweight Modular Honeypot for Network Attack Observation

Tatum Kelley
Electrical Engineering and Computer Science Department
Dayonta Beach, Florida
kelleyt1@my.erau.edu

Kellan Shew
Electrical Engineering and Computer Science Department
Dayonta Beach, Florida
kelleyt1@my.erau.edu

***Abstract*—This project presents the design and implementation of Catch-and-Log Honeypot, a lightweight and modular Phyton-based honeypot system. The system intended to capture automated scanning activity and low-skill intrusion attempts. The system also emulates common network services, logs attackers behavior with metadata, and provides analytics for observing attack patterns. It is designed for safe deployment using containerization and strict egress controls, the honeypot supports reproducible experimentation and foundational threat intelligence data collection.**

Keywords—honeypot, network security, intrusion detection, service emulation, threat analysis, automated scanning, attack logging, cybersecurity monitoring, Python

I. Introduction

Modern networks are regularly scanned and probed by automated bots searching for misconfigurations or vulnerable services. Honeypots provide a controlled way to observe such activity without risking the production systems. This project aims to develop a simple yet extensive honeypot that captures attacker interactions across emulated services such as HTTP, Telnet shells, and SSH like banners.

II. Background and Related Work

Honeypots vary from simple, low-interaction decoys to full system of high-interaction traps. Low-interaction honeypots emulate select protocols to safely collect behavioral data without exposing a full OS. This project situates itself in the low-interaction category as it prioritizes safety, modularity, and clarity over realism.

Existing platforms demonstrate the importance of:

- Restricting outbound connections.
- Isolating execution environments.
- Using consistent log schemas.
- Supporting multi-service emulation.

The Catch-and-Log Honeypot adopts these best practices all while remaining intentionally lightweight for educational and research purposes.

III. System Design

A. Architecture Overview

The honeypot systems is deployed as a lightweight Python application running multiple threaded services. Each service listens on its own port and logs interaction evens in a JSONL format. The following major components form the system:

1. Service Listeners – Python modules that accept inbound connections for HTTP and a Telnet/SSH-like banner service. These listeners display banners, accept user input, and return canned responses. No real system commands are executed.
2. Logging Pipeline – All interaction events (HTTP requests, login attempts, shell commands, and session closings) are recorded as individual JSON lines. Each entry includes:
    - Timestamp
    - Event type
    - Service
    - Source IP/ Source port
    - Destination port
    - Method/Path (HTTP)
    - Username/Password (Telnet)
    - Shell commands issued
    - Session duration
3. Storage – Events are appended to a structured JSONL log file. While no explicit schema validation is performed, the centralized logger ensures consistent fields across all services.
4. Analysis (Planned) – Logs can be used with external scripts to compute simple analytics such as:
    - Number of connections over time
    - Most attempted credentials
    - Most common HTTP paths
    - Most frequently used shell commands

The analysis tool is planned but not included in the core implementation.

5. Security Controls – The design emphasizes safe interaction.
    - VM isolation under VirtualBox
    - Egress-deny firewall policy
    - Docker sandboxing
    - Restricted user permissions
    - No real shell or system commands exposed

IV. Implementation

A. Programming Language and Environment

All services are implemented in Python to maximize readability and portability.
Deployment occurs on:

- VirtualBox Linux VM
- Ubuntu
- Docker
- VS Code server for remote editing

B. Service Emulation

The system implements two emulated services:

- Telnet-like login shell: Displays a text banner, accepts username/password input, logs the attempt, and drops the user into a simulated shell. The shell supports basic canned commands such as *help, ls, uname, whoami,* and *exit*. Unknown commands return "command not found."
- HTTP interface: Provides a minimal embedded-device style webpage at /, as well as a login page at */admin/login*. Both GET and POST requests are logged, including headers and submitted POST bodies. All login attempts intentionally fail.

No full SSH protocol or extended HTTP probe are implemented; the interaction is intentionally shallow, consistent with a low-interaction honeypot.

C. Logging Format

Each log entry is a single JSONL object. Example fields:

```
{
"ts": 1733278944.12,
"event_type": "login_attempt",
"service": "telnet",
"src_ip": "192.168.1.50",
"src_port": 50124,
"dst_port": 2323,
"username": "admin",
"password": "admin123",
"success": false
}
```

D. Deployment and Runtime

The honeypot runs as a Python application inside a Linux virtual machine. Each service executes inside its own thread and remains active until manually stopped. While the system is designed with containerization in mind, the currect implementation does not use Docker; instead, threaded services provide isolation at the application level.

V. Dataset Description

The dataset consists of JSONL formatted logs generated by the Telnet-like service and HTTP service. Each entry includes metadata such as:

- Timestamp
- Source IP and port
- Login attempts
- HTTP headers
- POST bodies
- Shell commands
- Session durations

VI. Analysis Methods

Although analysis scripts are not part of the core implementation, the structured logs enable post-processing to extract patterns such as:

1. Temporal Trends – Number of connections per hour/day.
2. Credential Attempts – Most frequently attempted usernames/passwords.
3. Probe Behavior – Most common HTTP paths or Telnet commands.
4. IP Distribution – Basic geographic estimation via public IP lookups APIs.
5. Service Popularity – Which emulated service receives the most attention

VII. Ethical and Safety Considerations

Because the system interacts with real external IP addresses, strict ethical controls are required:

- No counterattacks or outbound traffic
- Full isolation via VM containment and firewall rules
- Minimal emulation to prevent real system compromise
- Secure handling of collected IP data, used only for academic purposes

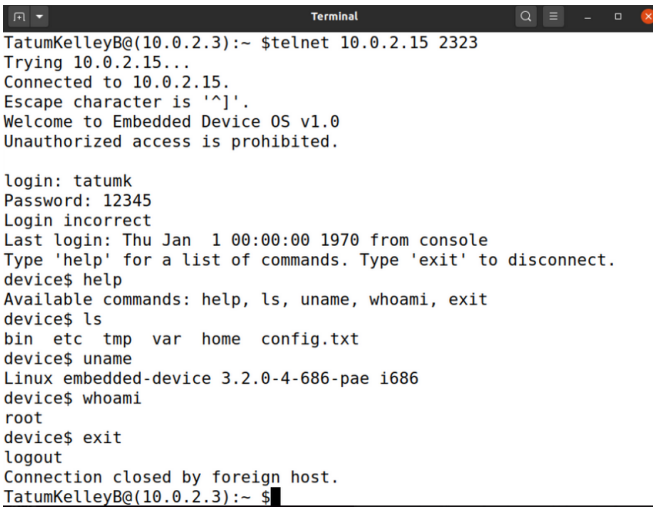
VIII. Timeline

The project follows the timeline from the proposal

1. Week 1 – Planning & setup
2. Week 2 – Core listeners
3. Week 3 – Service Emulation
4. Week 4 – Logging & Schema
5. Week 5 – Analysis Scripts
6. Week 6 – Data Collection
7. Week 7 – Final Report & Presentation

IX. Results

Started the honeypot system.



Attempting an attack on the server.

Data logged into the JSONL file obtained from the attackers inputs.

X. Conclusion

The Catch-and-Log Honeypot project successfully implements a controlled environment for observing real-world scanning and attack behavior. The system is lightweight, modular, and designed with strong isolation principles. Data collected during deployment enables meaningful insight into automated attacker behavior and provides a foundation for further security research, including improved dashboards, deeper analysis, or expansion into medium-interaction honeypots.

XI. References

[1] PyPI, “honeypots,” *Python Package Index*, Accessed: Dec. 2024. [Online]. Available: https://pypi.org/project/honeypots/

[2] T. Coy, “Creating a Honeypot,” *Medium*, 2023. Accessed: Dec. 2024. [Online]. Available: https://medium.com/@ecojumper30/creating-a-honeypot-f2b4cc33385a

[3] N. Y. Chan, “Build a Honeypot With Python,” *freeCodeCamp*, Feb. 2023. Accessed: Dec. 2024. [Online]. Available: https://www.freecodecamp.org/news/build-a-honeypot-with-python/