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
Daytona Beach, Florida
shewk1@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 on a Linux virtual machine running Docker containers. Each container exposes an emulated service and generates structured logs. The following major components form the system:

1. Service Listeners – Lightweight Python scripts that accept inbound connections (HTTP, Telnet, SSH like banners)

They display banners, accept basic input, and return canned responses.

2. Logging Pipeline – Each request or session is recorded as a JSONL event with fields including:
    - Timestamp
    - Service type
    - Source IP/ Source port
    - Destination port
    - Attempted credentials
    - Requested paths or payloads
    - Session duration
    - Optional geolocation enrichment
3. Storage and Schema Checking – Logs are validated via basic schema checks before being written to the disk.
4. Analysis Dashboard – A Python tool that reads logs and produces simple analytics:
    - Connections over time
    - Top attempted credentials
    - Most requested URLs
    - Most active IP address
5. Security Controls
    - 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 – Week-by-week implementation follows the project timeline:

- Initial listeners
- Telnet/SSH login prompts (fake)
- Minimal text-based shell with canned command outputs
- HTTP login page, common admin paths, and probe endpoints

C. Logging Format – Each log entry is a single JSONL object. Example fields:

<INSERT POSSIBLE OUTPUT HERE>

D. Deployment and Runtime – The VM runs the containerized honeypot continuously during our Week 6 timeline. Daily checkers ensure:

- Logs are growing
- Containers are running
- No unintended outbound connections occur

V. Dataset Description

The dataset consists of original attack traffic collected during deployment. Each event contains:

- Timestamp
- Service (HTTP/Telnet/SSH banners)
- Source IP and port
- Destination port
- Session duration
- Attempted credentials
- Response codes for HTTP

This dataset shows early behavior of scanners, botnets, credential-stuffing attempts, and misconfigured crawlers.

VI. Analysis Methods

Collected logs are processed using Python scripts to extract:

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

<ENTER DATA HERE LATER>

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

<ENTER DATA HERE LATER>