Unix Administration
Project Report

Honeypot

Université Saint-Joseph de Beyrouth
Faculté d'ingénierie
Institut national des télécommunications
et de l'informatique

November 10, 2025
Submitted by

Andrew Zgheib
andrew.zgheib@net.usj.edu.lb

Contents

Abstract

This project implements and analyzes three honeypot environments to capture and study real-world malicious behavior. Deployments include (1) T-Pot on Microsoft Azure for multi-sensor, containerized high-interaction logging, (2) Cowrie running on VMware as an SSH/Telnet low-to-medium interaction honeypot, and (3) a custom Python-based honeypot emulating SSH, FTP, and HTTP on a separate VMware VM. The report covers architecture, methodology, implementation details, results, and future work recommendations.

Chapter 1

Introduction

Honeypots are decoy systems purposely designed to be probed, attacked, or even compromised to collect intelligence on attackers' tools, techniques, and procedures (TTPs). They provide a safe environment for observing malicious activity without exposing production assets. This project evaluates three different approaches to honeypot deployment to measure coverage, data richness, operational complexity, and security risk.

1.1 Objectives

1. Deploy three honeypot solutions across cloud and virtualized environments.
2. Collect, parse, and analyze attack telemetry (connection attempts, brute-force credentials, commands, file uploads, etc.).
3. Compare strengths and weaknesses of a comprehensive platform (T-Pot), a specialized honeypot (Cowrie), and a custom low-interaction solution.
4. Produce recommendations and future work roadmap for research and production integration.

1.2 Categories

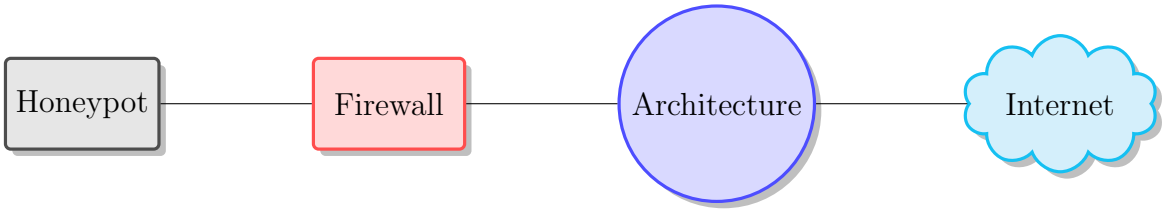
Key categories include:

- **Low-interaction honeypots**: Emulate services with limited functionality (easier to deploy, lower risk, limited intelligence).
- **High-interaction honeypots**: Full systems that allow extensive attacker engagement (richer telemetry, greater risk).
- **Hybrid honeypots**: Combine the best of both worlds to balance risk and data quality.

Chapter 2

Network Architecture

The overall network architecture applied across all deployments for this project can be summarized as seen below:



- **Honeypot Server**: Hosts the decoy service(s).
- **Firewall**: Controls and filters traffic to isolate honeypots from critical systems.
- **Architecture**: Placeholder network architecture tailored to specific needs.
- **Internet**: Honeypots are exposed to real-world (or simluated) attacks.

This architecture ensures attackers can interact with the honeypots while preventing the exploitation of actual network resources. Each honeypot runs in a segmented environment with strict controls. No outbound traffic is allowed except to designated analysis servers unless specifically required and sandboxed.

Chapter 3

Custom Honeypot VMware

3.1 Methodology

A minimal custom honeypot that emulate service banners and record requests (SSH, FTP, HTTP) using Paramiko and sockets. It is lightweight, fully modular, and fast to prototype and tailor.

3.2 Implementation

3.2.1 Virtual Machine Configuration

A virtual machine was configured to use `Ubuntu Server LTS` with adequate compute and storage resources (`2 vCPUs, 4 GiB memory`) and a `20 GB disk` to support the resource demands of the custom honeypot.

3.2.2 Security Hardening

The VM's security hardening process included configuring the Uncomplicated Firewall (UFW) to enforce a strict "deny by default" policy. Only essential network ports were explicitly allowed.

3.2.3 Honeypot Architecture

Below is a compact diagram of the main runtime components and how they interact:

- External actor (attacker) connecting to the service.
- A listener handled by the main entrypoint (`main.py`).
- A dispatcher that delegates to protocol handlers located in the `/protocols`.
- A central `logger.py` used by handlers to record events.

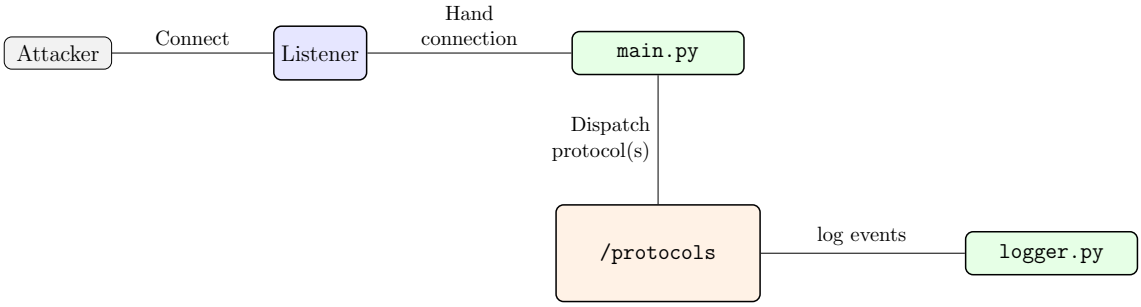


Figure 3.1: High-level architecture of the custom honeypot

3.3 Future Work

While the current implementation provides a lightweight, modular honeypot capable of simulating SSH, FTP, and HTTP services, there are several avenues to extend its capabilities, improve realism, and enhance security research value.

3.3.1 Support for more protocols

Leveraging the honeypot's modular approach, additional protocols commonly targeted by attackers can be implemented, such as:

- **Email Protocols**: Protocols like SMTP and IMAP can be implemented to capture email-related attacks or credential harvesting attempts.
- **Database Protocols**: Database protocols related to MySQL, PostgreSQL, MongoDB, or other databases, can be implemented to detect attacks targeting exposed databases.

3.3.2 Machine Learning and Threat Detection

To make the honeypot even more advanced, we can apply machine learning tools to honeypot logs in order to detect novel attack patterns or automated scanning attempts. The attacks can be classified by developing classification models based on protocol, behavior, or payload type.

Chapter 4

Cowrie VMware

4.1 Methodology

Cowrie was deployed as a focused SSH/Telnet honeypot in an isolated VMware virtual machine to collect simulated session logs and attacker commands with medium interaction.

4.2 Implementation

4.2.1 Virtual Machine Configuration

The VMware VM was configured to use the same configuration as the custom honeypot.

4.2.2 Software Installation

First, system-wide support for Python virtual environments and other dependencies were installed. Then, to run Cowrie with a dedicated non-root user id, a user `cowrie` was created with the `--disabled-password` flag. After that, the Cowrie GitHub Repository was cloned and a python virtual environment was setup to install the required packages.

4.2.3 Configuration

The configurations applied to the original honeypot were:

- Specifying the allowed credentials by creating a custom `/cowrie/etc/userdb.txt` file
- Adding custom files in directories by running `fsctl /share/cowrie/fs.pickle` and running commands inside the `fsctl` environment like `touch`, `mkdir`, and `chmod`.

4.2.4 PotsgreSQL & VirusTotal Integrations

Utilizing Cowrie's modular approach, PostgreSQL was setup by downloading the `psycopg2` python library in Cowrie's virtual environment, creating a `cowrie` user account with specific access privileges on a `cowrie` table, and enabling the PotsgreSQL plugin in `/cowrie/etc/cowrie.cfg`.

Additionally, VirusTotal's v3 API was integrated to automatically scan and track malicious files and URLs captured by the honeypot. This was done by enabling the VirusTotal plugin in `/cowrie/etc/cowrie.cfg` and providing it the API key.

Chapter 5

T-Pot Azure

5.1 Methodology

T-Pot was used as an integrated honeypot stack to maximize sensor coverage with minimal manual glue work. T-Pot packages many honeypots (ElasticHoney, Cowrie, Dionaea, Honeytrap, Glastopf, etc.) into Docker containers with a centralized ELK-based logging stack and dashboards. It is deployed publicly in an Azure VM to observe diverse global automated activity.

5.2 Implementation

5.2.1 Virtual Machine Configuration

The Azure Virtual Machine was configure to use `Ubuntu LTS` with adequate compute and storage resources (`Standard D4s v3 (4 vCPUs, 16 GiB memory)`) and a `128 GB disk` to support the resource demands of T-Pot and the ELK stack. It is hosted in the `France Central (Zone 3)` region due to its close proximity.

5.2.2 Security Hardening

The VM was also hardened by restricting SSH and web portal access to specific ports, with SSH authentication being strictly through certificates and keys (no username/password authentication).

5.2.3 Software Installation

T-Pot was installed using the official script hosted on its GitHub Repository.

5.2.4 Network Configuration

The Azure Network Security Group (NSG) was configured to permit inbound traffic on necessary honeypot sensor ports, including, but not limited to, 22, 80, 443, 21, 23, and 8080.

5.3 Results

Within the first 3 days, **371K** of automated connection attempts were observed. **270K** of which where from a DDoS attack.

The most common activities where:

1. SSH brute-force attempts from multiple autonomous systems (botnets). Many attempts used common username:password combos (e.g., `root:123456`, `admin:admin`, `test:123`, etc.).
2. Automated malware download attempts via common `wget/curl` payloads.
3. Nmap scanning attacks and other network exploration tools.

The ELK dashboards provided aggregated views, geo IP mapping, timeline of peaks, and much more visualizations enabling a great foundation for studying modern attacker behaviors.

5.4 Future Work

The current honeypot implementation provides a flexible and modular framework for simulating common network services and capturing attacker interactions. However, there is significant potential to extend its capabilities.

5.4.1 Dynamic Alerting

Deploy real-time alerts via email or other tools for all high-severity activities, including successful file uploads and detected reverse-shell attempts. Each alert should include contextual metadata and links to relevant incident response playbooks to accelerate triage and containment.

5.4.2 Automated Malware Detonation

Integrate an automated malware detonation pipeline to handle uploaded binaries. The pipeline will sandbox files, extract indicators of compromise, and feed structured results into the SIEM and incident management systems for automated correlation and reporting.

Chapter 6

Conclusion

The three-honeypot deployment provided complementary views into attacker activity: T-Pot offered wide coverage and rich dashboards, while Cowrie and the custom Python honeypot delivered rapid and detailed session transcripts, with tailored visibility into specific probes.