# Automated Honeypot Monitoring & Threat Analysis

**Prepared by:** Abdulaziz Al Saadat

## Table of Contents

# 1. Project Overview

The objective of this project is to deploy a Honeypot (Artillery) to act as a decoy for attackers and automate the process of analyzing their intent. By deploying a deceptionbased network decoy, the system captures malicious activity that would otherwise remain undetected, and processes it through a fully automated analysis pipeline.

# 2. Technical Architecture

The system operates through a specialized workflow designed for deception and automated analysis.

## 2.1 Infrastructure & Environment

To ensure a secure and isolated testing environment, the project was deployed as follows:

- **Virtualization:** Hosted on VMware Workstation to maintain network isolation from the host machine.
- **Operating System:** Powered by Ubuntu 22.04 LTS, providing a stable Linux environment for the honeypot services and automation engine.

## 2.2 Core Components

- **Deception Layer (Artillery Honeypot):** Configured to open decoy ports and monitor sensitive system files to attract and observe malicious interaction attempts.
- **Detection & Parsing:** A custom **Python** script monitors the Honeypot's logs for interaction signatures (e.g., Port Scanning or Brute Force), enabling real-time detection of hostile behavior without manual log inspection.
- **OSINT Enrichment:** Every "hit" captured by the script is automatically crossreferenced with the **VirusTotal API** and Geo-location databases to evaluate the attacker's threat level.
- **Instant Alerting:** Final analysis results are delivered via a **Telegram Bot**, providing a clear, immediate report on the attacker's identity and reputation.

## 3. Implementation Logic

The core of the project is the Log-Watchdog script. Instead of manually reviewing thousands of lines of log data, the script:

- Identifies Attacks: Searches for predefined attack indicators such as attack, blocked, and brute-force related patterns.
- Filters Noise: Ignores system-level configuration messages to prevent alert fatigue.
- Extracts Metadata: Uses Regex to pinpoint the attacker's IP address for further investigation.

The following diagram illustrates the end-to-end workflow of the automated Honeypot monitoring system.

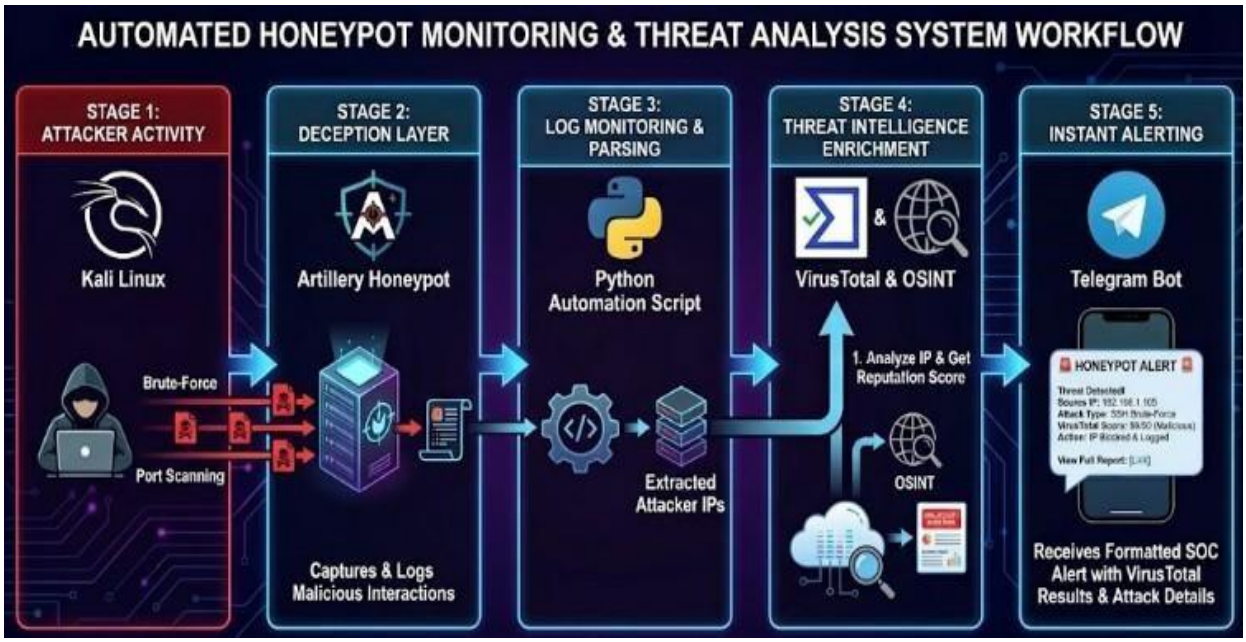


*Figure 1-HoneyPot WorkFlow Diagram*

## 4. Validation via Attack Simulation

To test the Honeypot's effectiveness, a simulated attack was performed from a Kali Linux machine:

- Attack Phase: Using Hydra, multiple SSH connection attempts were made against the Honeypot's decoy port.
- Honeypot Reaction: Artillery detected the "Brute Force" pattern and immediately logged the source IP.
- Automated Alert: The SOC bot captured the log, performed a reputation check on VirusTotal, and sent a detailed alert stating exactly how many vendors have flagged this specific attacker.

This simulation confirms the system's capability to detect, analyze, and report real-world attack behavior in an automated SOC-like workflow.

In addition to live attack simulation, a controlled validation technique was performed to further verify the effectiveness of the monitoring pipeline. Known malicious IP addresses with poor reputation were manually researched using public OSINT sources and threat intelligence platforms.

These IP addresses were then manually injected into the Honeypot log files to simulate real-world attack entries. This approach ensured that the detection, parsing, enrichment, and alerting mechanisms function correctly even when processing historical or externally sourced threat data.

The system successfully identified the injected IP addresses, enriched them with reputation data, and generated alerts, confirming the reliability and accuracy of the automated analysis workflow.

Together, both live attack simulation and controlled log injection validate the system's ability to reliably detect and analyze malicious activity under realistic and controlled conditions.

## 5.Key Outcomes

- Proactive Defense: Using a Honeypot allows for the early detection of reconnaissance efforts.
- Improved Visibility: Real-time Telegram alerts ensure that any interaction with the "trap" is noticed instantly.
- Automated Intelligence: The manual task of checking IP reputations is now fully automated via API integration.

## 6. Summary & Lessons Learned

Overall, this project demonstrates how deception technologies combined with automation and threat intelligence can significantly enhance security monitoring capabilities while reducing analyst workload.

Key takeaways from this project include:

- **Log Parsing Efficiency:** Successfully implemented Regex patterns to extract critical data from complex security logs.
- **Real-time Awareness:** Reduced the time-to-alert from hours (manual review) to seconds (automated push).
- **Noise Management:** Learned how to filter out non-essential system messages to maintain focus on actual threats.