Dynamic Honeypot Orchestrator with Web Dashboard

Devrajsinh Shripalsinh Jhala
h20240120@goa.bits-pilani.ac.in
Birla Institute of Technology and Sciences, Pilani

Suryadevsinh Harishchandrasinh Zala
h20240127@goa.bits-pilani.ac.in
Birla Institute of Technology and Sciences, Pilani

Udit Chaudhary
h20240021@goa.bits-pilani.ac.in
Birla Institute of Technology and Sciences, Pilani

Gladwin Kurian
h20240040@goa.bits-pilani.ac.in
Birla Institute of Technology and Sciences, Pilani

ABSTRACT

Deploying, managing, and extracting meaningful insights from network honeypots often involves significant manual effort and fragmented tooling. This paper introduces the Honeypot Orchestrator, a comprehensive web application designed to streamline the entire honeypot lifecycle. The system integrates a high-performance FastAPI backend with an interactive Next.js frontend dashboard, enabling users to easily configure and deploy various honeypot types (SSH, Web) as Docker containers. It automatically aggregates attack data, provides real-time attack monitoring, and offers intuitive data visualization through charts and tables. Key contributions include sophisticated attack simulation capabilities with varying complexity levels for testing and demonstration purposes, and integration with Google's Gemini AI for automated threat analysis and security recommendations. This unified platform aims to significantly reduce the operational overhead associated with honeypot management and enhance threat intelligence gathering for cybersecurity professionals and researchers.

CCS CONCEPTS

• **Security and privacy** → **Intrusion detection systems**; **Network security**; • **Software and its engineering** → *Application specific development environments*; • **Computing methodologies** → Natural language processing.

KEYWORDS

Honeypot, Orchestration, Network Security, Docker, Attack Monitoring, Attack Simulation, FastAPI, Next.js, Threat Intelligence, AI Analysis

1 INTRODUCTION

Network honeypots are invaluable tools in cybersecurity, serving as decoys to attract and analyze malicious activities. They provide crucial insights into emerging threats, attacker methodologies, and potential vulnerabilities. However, managing a fleet of honeypots, especially diverse types, presents significant operational challenges. Deployment complexity, log aggregation, real-time monitoring, data correlation, and effective analysis often require specialized knowledge and considerable manual intervention, hindering the efficient extraction of threat intelligence.

The Honeypot Orchestrator project directly addresses these challenges by providing a unified, web-based platform designed to simplify and automate the entire honeypot management lifecycle. Our primary objective is to empower cybersecurity professionals, researchers, and enthusiasts with an intuitive system that lowers the barrier to entry for effective honeypot utilization.

The system's core contributions are:

- **Simplified Deployment and Management:** Abstracting Docker complexities for easy deployment and removal of common honeypot services (SSH, Web) via a graphical interface.
- **Centralized Monitoring and Data Aggregation:** Automatically collecting attack logs from all deployed honeypots into a single, persistent store, eliminating the need for manual log retrieval.
- **Enhanced Testing Capabilities:** Offering sophisticated attack simulation features with adjustable complexity and types to validate honeypot effectiveness and security posture.
- **AI-Driven Insights:** Integrating with Google's Gemini AI to provide automated analysis of attack patterns and actionable security recommendations, reducing analysis time.

The architecture comprises a Python-based FastAPI backend handling core logic, data persistence, Docker orchestration, and external API interactions, coupled with a modern Next.js frontend providing an interactive dashboard for users. This separation ensures scalability and maintainability while delivering a responsive user experience.

2 TECHNOLOGY STACK

- **Backend:** Python with the FastAPI framework for the API, interacting with Docker via `docker-py`. Data is stored in JSON files. Google Gemini API is used for AI features.
- **Frontend:** TypeScript with the Next.js/React framework for the user interface. Tailwind CSS is used for styling. Chart.js enables data visualization, and Lucide React provides iconography.
- **Core Concepts:** Asynchronous programming, RESTful API design, containerization (Docker).

3 FEATURES IMPLEMENTED

The Honeypot Orchestrator provides intuitive web-based management (create, view, delete) and simplified Docker deployment for standard honeypots (SSH, Web), including state recovery. Automated threat collection with deduplication ensures data integrity, while real-time visibility is provided through comprehensive dashboards, timelines, and statistical breakdowns (IST timestamps). Its

realistic testing environment allows simulating diverse network attacks (Login Brute-Force, SQLi, XSS, Port Scan, DoS, etc.) with adjustable complexity for validation. Furthermore, integrated Google Gemini AI offers intelligent insights via automated pattern analysis and tailored security recommendations, all accessed through a responsive Next.js/Tailwind CSS user interface.

4 METHODOLOGY

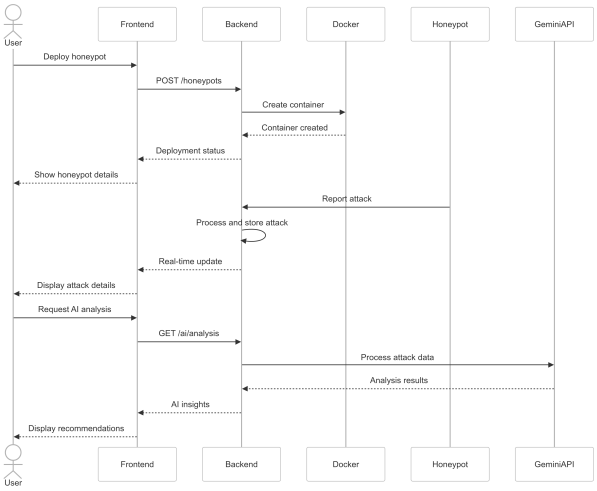

Figure 1: Data flow diagram of the Dynamic Honeypot Orchestrator Project

- **Docker Orchestration:** Abstracts Docker API calls via `docker-py` to manage the container lifecycle based on user actions and stored configurations. Dynamically assigns ports and uses labels for identification.
- **Attack Data Collection (Log Polling):** An asynchronous background task periodically fetches logs from running containers. This approach avoids needing agents inside honeypots but introduces a slight delay in detection compared to real-time methods. Parsing relies on predefined regular expressions and logic specific to known log formats.
- **Simulation Generation:** Uses rule-based generation with randomness. For single complex attacks, specific payloads and parameters are chosen from predefined pools based on the selected complexity level. For campaigns, it focuses on varying credentials for SSH login attempts based on complexity pools and timed execution via `asyncio.sleep`.
- **AI Interaction:** Providing summarized attack data or honeypot details and instructing the AI to perform analysis or generate recommendations in a specific format.
- **Persistence (File-Based):** Utilizes simple JSON file I/O for storing application state (honeypots, attacks). This is straightforward but lacks transactional integrity and scalability compared to traditional databases.

5 SOFTWARE ARCHITECTURE

The Honeypot Orchestrator is built upon a decoupled Client-Server architecture, promoting modularity and separation of concerns.

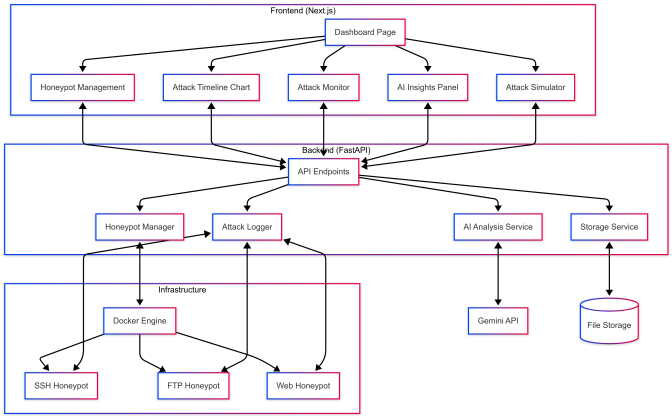

Figure 2: Overall Layered Architecture of the Dynamic Honeypot Orchestrator Project

The **Backend**, developed using Python and the FastAPI framework, serves as the central control plane and data processing engine. It exposes a RESTful API for all management and data retrieval operations requested by the frontend. Its key responsibilities include processing incoming API requests, managing honeypot configurations, orchestrating Docker container lifecycles (deployment, termination, log retrieval) via the Docker API, persisting application state (honeypot details, attack records) using a simple file-based JSON storage mechanism, interacting with the external Google Gemini API for AI analysis, and managing connections to push real-time attack data to clients. Background tasks handle periodic operations like synchronizing attack logs from containers.

The **Frontend**, a modern web application built with Next.js and React (TypeScript), provides the graphical user interface (GUI) for users. It presents dashboards, lists, forms, and visualizations. It interacts with the backend exclusively through the defined REST API for fetching data (e.g., honeypot lists, attack statistics) and initiating actions (e.g., deploying a honeypot, starting a simulation). Crucially, it also establishes and maintains a connection to the backend, allowing it to receive and display attack information in near real-time without constant polling. State management is handled primarily within React components using standard hooks.

This architecture clearly separates the presentation logic (frontend) from the core business logic, orchestration, and data handling (backend), connected via well-defined API contracts.

6 RESULTS

6.1 Honeypot Deployment

The system successfully allows users to define honeypot configurations and deploy them as Docker containers (`cowrie/cowrie`, `vulnerables/web-dvwa`) through the web interface. The backend correctly interacts with the Docker API to manage container lifecycles and recovers state effectively after restarts.

6.2 Attack Detection and Visualization

Attack events are effectively collected from active honeypots via log polling, parsed, deduplicated using hashing, and stored. The frontend dashboard provides near real-time updates in a detailed table with localized timestamps and informative visuals. Clear graphical summaries are provided through various visualization components, including timeline charts, daily bar graphs, distribution graphs, and statistical summaries, illustrating attack frequency, daily trends, type distribution, and common targets.

6.3 Attack Simulation

Users can successfully generate simulated attacks for testing. The "Test Single Attack" feature (`AttackSimulator`) creates varied attacks (SQLi, XSS, DoS, etc.) with complexity-dependent details via the `/attack-sim` API. The "Start Simulation" feature initiates sustained SSH brute-force campaigns, respecting configured parameters, with UI progress tracking.

6.4 AI-Powered Analysis

Integration with Google Gemini (`AIInsightsPanel`) provides valuable automated intelligence. The backend successfully summarizes data, prompts the AI, and the frontend displays the resulting textual analysis and security recommendations using Markdown, offering actionable insights with reduced manual effort.

6.5 User Experience and Performance

The Next.js frontend offers a responsive and interactive experience with a clean, modern interface styled using Tailwind CSS. Real-time updates and clear visual components enhance monitoring. Loading states improve perceived performance.

7 OPERATIONAL BENEFITS

The Honeypot Orchestrator offers significant practical advantages for users and organizations:

- **Reduced Operational Overhead:** Centralizes honeypot deployment, configuration, and monitoring, drastically reducing the time and effort required compared to managing individual instances manually.
- **Increased Efficiency:** Automates log collection, parsing, and deduplication, freeing up security personnel to focus on higher-level analysis and response rather than data wrangling.
- **Improved Threat Visibility:** Provides a unified, real-time view of attack activity across multiple honeypots, enabling faster identification of coordinated attacks or widespread campaigns targeting specific services.
- **Enhanced Situational Awareness:** Visualizations and statistics offer quick insights into attack trends, common vectors (e.g., prevalent attack types, targeted credentials), and most active sources, improving understanding of the current threat landscape targeting the monitored assets.
- **Accelerated Analysis:** AI-driven analysis and recommendations provide rapid initial assessments and potential mitigation steps, significantly speeding up the threat intelligence lifecycle.
- **Simplified Testing and Validation:** The integrated attack simulator allows for easy and repeatable testing of honeypot effectiveness, detection rules, and security configurations without requiring external attack tools.
- **Lowered Barrier to Entry:** The user-friendly web interface makes honeypot deployment and basic monitoring accessible even to users with limited Docker or command-line expertise.
- **Cost-Effective Monitoring:** Leverages open-source honeypot software and provides a centralized management layer, potentially reducing the need for multiple disparate commercial monitoring solutions.

8 LITERATURE REVIEW

Existing honeypot management solutions often struggle with complex deployment processes, limited visualization capabilities, and challenges in real-time monitoring [1]. While traditional solutions like Honeyd [4] require manual configuration and maintenance, leading to increased operational overhead and potential security risks, our Dynamic Honeypot Orchestrator addresses these limitations through automated Docker-based deployment and an intuitive web dashboard. Unlike existing tools such as HoneyPy [6] that often lack comprehensive visualization, our solution combines real-time attack monitoring with AI-powered analysis through Gemini API integration, providing actionable threat intelligence without requiring extensive security expertise. The system's containerized approach ensures proper isolation of honeypots while maintaining ease of deployment, addressing a critical security concern in high-interaction honeypots identified by [3]. Furthermore, our implementation's web-based dashboard with real-time visualization and attack simulation capabilities offers superior usability compared to traditional command-line interfaces like Modern Honey Network [5], making sophisticated honeypot management accessible to a broader range of security professionals while maintaining robust security features through container isolation and automated orchestration [2].

9 CONCLUSION

The Honeypot Orchestrator successfully integrates a FastAPI backend and Next.js frontend, creating a powerful, user-friendly platform that simplifies honeypot deployment via Docker, centralizes monitoring, and offers valuable insights through visualization (Chart.js) and AI analysis (Google Gemini). Featuring robust attack simulation for testing, the system provides a solid, extensible foundation for efficient honeypot operations and threat intelligence gathering. While the file-based JSON database limits scalability, future work includes adopting a more robust database, expanding honeypot support, implementing RBAC, and integrating with SIEM systems, building upon the project's valuable contribution to cybersecurity monitoring.

REFERENCES

[1] Wenjun Fan, Zhihui Du, David Fernández, and Víctor A. Villagrá. 2018. Enabling an Anatomic View to Investigate Honeypot Systems: A Survey. *IEEE Systems Journal* 12, 4 (2018), 3906–3919. https://doi.org/10.1109/JSYST.2017.2762161

[2] Daniel Fraunholz, Marc Zimmermann, Hans D. Schotten, and Daniel Schneider. 2018. A Security Evaluation of Modern Honeypots. *Journal of Cyber Security and Mobility* 7, 1 (2018), 87–116. https://doi.org/10.13052/jcsm2245-1439.714

[3] Marcin Nawrocki, Matthias Wahlisch, Thomas C. Schmidt, Christian Keil, and Jochen Schonfelder. 2016. A Survey on Honeypot Software and Data Analysis. *arXiv preprint arXiv:1608.06249* (2016).
[4] Niels Provos. 2004. A Virtual Honeypot Framework. In *Proceedings of the 13th USENIX Security Symposium*, Vol. 173. USENIX Association, San Diego, CA, 1–14.
[5] ThreatStream. 2020. Modern Honey Network. GitHub Repository. https://github.com/threatstream/mhn.
[6] Craig Valli, Priya Rabadia, and Andrew Woodward. 2017. Patterns and Patter - An Investigation into SSH Activity Using Kippo Honeypots. In *Proceedings of the 15th Australian Digital Forensics Conference*. Edith Cowan University, Perth, Western Australia.