International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

# Honeytrap: Ai-Powered Honeypot for Cyberattack Detection

PROJECT PHASE II

*submitted in partial fulfilment of the requirements*

*for the award of the degree in*

**MASTER OF TECHNOLOGY**

**in**

**CYBER FORENICS AND INFORMATION SECURITY**

by

**PARASU RAMAN M (241572101005)**

**DEPARTMENT OF CYBER SECURITY**

**JUNE 2026**



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

## DECLARATION

I **PARASU RAMAN M** (Reg.No) **241572101005**, hereby declare that the Project Report entitled "**HONEYTRAP: AI-POWERED HONEYPOT FOR CYBERATTACK DETECTION**" is done by me under the guidance of **DR. V.N. RAJAVARMAN** is submitted in partial fulfilment of the requirements for the award of the degree in MASTER OF TECHNOLOGY in CYBER FORENICS AND INFORMATION SECURITY.

**DATE:**

**PLACE:** **SIGNATURE OF THE CANDIDATE**



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

## DEPARTMENT OF CYBER SECURITY

## BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of Mr. **PARASU RAMAN M** Reg. No **241572101005**, who carried out the project entitled "**HONEYTRAP: AI-POWERED HONEYPOT FOR CYBERATTACK DETECTION**" under our supervision from Jan 2026 to Jun 2026.

| **Internal Guide** | **Project Coordinator** | **Department Head** |
|---|---|---|
| **DR. V.N. Rajavarman** | **Dr. S. Geetha & Dr. B. Raja** | **Dr. P. Dinesh Kumar** |
| Professor | Professors | Associate Professor |
| Department of CSE | Department of CSE | Department Of Cyber Security |
| Dr. M.G.R. Educational and Research Institute | Dr. M.G.R. Educational and Research Institute | Dr. M.G.R. Educational and Research Institute |

**Submitted for Viva Voce Examination held on** ________________

**Internal Examiner** **External Examine**



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

## ACKNOWLEDGEMENT

I would first like to thank our beloved Chancellor **Thiru. Dr. A.C. Shanmugam, B.A., B.L.**, President **Er. A.C.S. Arunkumar, B.Tech.,** and Secretary **Thiru A. Ravikumar** for all the encouragement and support extended to us during the tenure of this project and also our years of studies in this wonderful University.

I express our heartfelt thanks to our Vice Chancellor **Dr. S. Geethalakshmi** in providing all the support for my Project.

I express our heartfelt thanks to our Dean & Head of the Department, **Prof. Dr. S. Geetha**, who has been actively involved and very influential from the start till the completion of my project.

I express our heartfelt thanks to our Head of the Department, **Dr. P. Dinesh**, who has been actively involved and very influential from the start till the completion of my project

Our sincere thanks to our Project Coordinators **Prof. Dr. S. Geetha, Dr. B. Raja** and Project guide **Dr. V.N. Rajavarman** for their continuous guidance and encouragement throughout this work, which has made the project a success.

I would also like to thank all the teaching and non-teaching staffs of Computer Science and Engineering department, for their constant support and the encouragement given to us while we went about to achieving my project goals.



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

# CONTENTS

| CHAPTER NO | TITLE | PAGE NO |
|---|---|---|
| | **LIST OF FIGURES** | **iv** |
| | **LIST OF TABLES** | **v** |
| | **LIST OF ABBREVIATION** | **vi** |
| | **ABSTRACT** | **1** |
| **1** | **INTRODUCTION** | **2** |
| | 1.1 BACKGROUND | 2 |
| | 1.2 PARADIGM OF CYBER DECEPTION AND HONEYPOTS | 4 |
| | 1.3 PROBLEM STATEMENT | 6 |
| | 1.4 OBJECTIVES OF THE PROJECT | 8 |
| | 1.5 SCOPE AND LIMITATIONS | 9 |
| | 1.6 ORGANIZATION | 11 |
| **2** | **LITERATURE SURVEY** | **13** |
| | 2.1 INTRODUCTION | 13 |
| | 2.2 THE EVOLUTION OF HONEYPOT ARCHITECTURES | 14 |
| | 2.2.1 LOW-INTERACTION HONEYPOTS | 14 |
| | 2.2.2 HIGH-INTERACTION HONEYPOTS | 15 |
| | 2.2.3 THE MODERN HYBRID APPROACH | 16 |
| | 2.3 REAL-TIME THREAT VISUALIZATION AND EVENT-DRIVEN ARCHITECTURE | 16 |
| | 2.4 ARTIFICIAL INTELLIGENCE IN BEHAVIORAL THREAT CLASSIFICATION | 18 |
| | 2.5 BLOCKCHAIN TECHNOLOGY FOR FORENSIC DATA INTEGRITY | 19 |
| | 2.6 SUMMARY OF LITERATURE AND GAP ANALYSIS | 20 |


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

| | | |
|---|---|---|
| **3** | **REQUIREMENT SPECIFICATION** | **21** |
| | 3.1 INTRODUCTION | 21 |
| | 3.2 OVERALL SYSTEM DESCRIPTION | 21 |
| | 3.2.1 PRODUCT PERSPECTIVE | 21 |
| | 3.2.2 USER CLASSES AND CHARACTERISTICS | 22 |
| | 3.2.3 OPERATING ENVIRONMENT | 23 |
| | 3.3 FUNCTIONAL REQUIREMENTS | 23 |
| | 3.4 NON-FUNCTIONAL REQUIREMENTS | 26 |
| | 3.5 SYSTEM REQUIREMENTS | 28 |
| | 3.5.1 SOFTWARE REQUIREMENTS | 28 |
| | 3.5.2 HARDWARE REQUIREMENTS | 29 |
| **4** | **DESIGN** | **30** |
| | 4.1 INTRODUCTION | 30 |
| | 4.2 OVERALL SYSTEM ARCHITECTURE | 30 |
| | 4.2.1 TIER 1: THE DECEPTION LAYER | 31 |
| | 4.2.2 TIER 2: CORE LOGIC AND PROCESSING | 32 |
| | 4.2.3 TIER 3: VISUALIZATION AND CONTROL | 32 |
| | 4.3 MODULE DESIGN | 33 |
| | 4.3.1 MODULE 1: HIGH-FIDELITY DECEPTION ENGINE | 33 |
| | 4.3.2 MODULE 2: FORENSIC AND GEOLOCATION ANALYZER | 33 |
| | 4.3.3 MODULE 3: AI-POWERED AUTONOMOUS RESPONSE | 34 |
| | 4.3.4 MODULE 4: SECURE BLOCKCHAIN PERSISTENCE | 34 |
| | 4.4 DATA FLOW AND SEQUENCE DESIGN | 35 |
| | 4.5 UML CLASS DIAGRAM | 36 |
| | 4.6 USER INTERFACE (GUI) DESIGN | 37 |
| | 4.7 SECURITY AND ISOLATION DESIGN | 39 |


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

| | | |
|---|---|---|
| **5** | **IMPLEMENTATION** | **41** |
| | 5.1 INTRODUCTION | 41 |
| | 5.2 DEVELOPMENT AND TESTING ENVIRONMENT | 41 |
| | 5.3 PROCEDURE TO IMPLEMENT | 42 |
| | 5.4 CHALLENGES AND SOLUTIONS | 43 |
| | 5.5 TESTING APPROACH | 43 |
| | 5.6 RESULTS | 44 |
| | 5.7 DISCUSSION | 46 |
| | 5.8 COMPARISON WITH EXISTING TOOLS | 46 |
| | 5.9 VALIDATION | 48 |
| **6** | **CONCLUSION** | **49** |
| | 6.1 SUMMARY | 49 |
| | 6.2 ACHIEVEMENTS | 50 |
| | 6.3 LIMITATIONS | 51 |
| | 6.4 FUTURE WORK | 52 |
| | 6.5 CLOSING REMARKS | 53 |
| | **REFERENCES** | **54** |
| | **APPENDIX** | **56** |


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

# LIST OF FIGURES


| FIGURE NO. | FIGURE NAME | PAGE NO. |
|---|---|---|
| 4.1 | SYSTEM ARCHITECTURE | 31 |
| 4.2 | MODULE DESIGN | 33 |
| 4.3 | UML CLASS DIAGRAM | 36 |
| 5.1 | RESULT: IMPLEMENTATION | 45 |
| 5.2 | RESULT: TRAP-AWS | 45 |
| 5.3 | RESULT: TRAP-MICROSOFT | 45 |
| 5.4 | RESULT: DASHBOARD | 46 |
| 5.5 | RESULT: REPORT | 46 |



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

# LIST OF TABLES


| TABLE NO. | TABLE NAME | PAGE NO. |
|---|---|---|
| 5.1 | COMPARISON | 47 |



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

## LIST OF ABBREVIATION

| | | |
|---|---|---|
| 1 | AI | Artificial Intelligence |
| 2 | API | Application Programming Interface |
| 3 | DDoS | Distributed Denial of Service |
| 4 | FTP | File Transfer Protocol |
| 5 | GUI | Graphical User Interface |
| 6 | HTTP | Hypertext Transfer Protocol |
| 7 | IP | Internet Protocol |
| 8 | JSON | JavaScript Object Notation |
| 9 | SOC | Security Operations Center |
| 10 | SSH | Secure Shell |


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

## ABSTRACT

As cyber threats become increasingly automated and sophisticated, traditional reactive security measures are no longer sufficient to protect critical infrastructure. To address this gap, this project introduces Honeytrap, a proactive, AI-powered honeypot system designed to deceive attackers, extract forensic intelligence, and autonomously neutralize threats in real time. The system utilizes a multi-tiered architecture that seamlessly integrates high-fidelity deception, machine learning, and decentralized ledger technology to disrupt the attack lifecycle. The foundational layer of the Honeytrap framework features a highly realistic deception engine that deploys traps across commonly targeted protocols, including HTTP, FTP, SSH, and SQL. By hosting simulated environments such as enterprise login portals, the system actively lures threat actors. Upon interaction, the core logic immediately executes forensic extraction, capturing the attacker's true IP address, stripping network routing wrappers, and resolving their precise geographical location via an asynchronous API. This intelligence is instantly streamed to a Live Forensic Dashboard, providing administrators with a zero-latency interactive threat map and analytics in under 120 milliseconds.

To elevate the system from a passive monitoring tool to an active defense mechanism, the architecture incorporates an artificial intelligence microservice. This machine-learning classifier evaluates connection behaviors and payload signatures to accurately distinguish automated botnets from human hackers. When a malicious bot is identified, the system autonomously deploys a 'Kill Switch' to sever the connection, drastically reducing incident response times without requiring manual intervention. Furthermore, to guarantee the integrity and court admissibility of the gathered evidence, Honeytrap utilizes secure blockchain persistence. By writing the forensic logs directly to an immutable decentralized ledger, the system ensures the data remains entirely tamper-proof. Initial deployment results demonstrate a 100% credential capture rate alongside highly efficient resource utilization. Ultimately, Honeytrap provides a resilient, scalable, and intelligent solution for modern digital forensics, transforming network defense through proactive deception and automated response.

**Keywords:** Cybersecurity, AI-Powered Honeypots, Digital Forensics, Real-Time Threat Detection, Blockchain Persistence, Autonomous Defense, Network Deception.



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

# CHAPTER 1

# INTRODUCTION

## 1.1 BACKGROUND

The fundamental architecture of the global technological ecosystem has undergone a radical and irreversible transformation over the past two decades. In the early days of computing, digital networks were largely localized, physical entities. Organizations operated within closed, easily definable perimeters, often referred to as intranets. Protecting these networks was a straightforward matter of securing the physical endpoints and establishing a strong perimeter gateway. However, the modern digital landscape bears little resemblance to these early networks. We have transitioned into an era of hyper-connectivity, driven by the ubiquitous adoption of cloud computing, the proliferation of the Internet of Things (IoT), mobile computing, and decentralized infrastructure. While this digital paradigm shift has enabled unprecedented levels of global communication, remote accessibility, and operational efficiency, it has concurrently expanded the digital attack surface to an unmanageable scale. Today, every connected smart device, exposed server port, third-party API integration, and cloud-hosted web application represents a potential entry point into an organization's most critical and sensitive infrastructure.

Parallel to this evolution in networking, the nature of cyber threats has transformed drastically. In the late 1990s and early 2000s, cyberattacks were frequently manual, slow-moving operations, often executed by individual actors seeking notoriety, testing their technical skills, or attempting minor financial theft. These early hackers manually probed networks, looking for specific vulnerabilities to exploit. Today, the landscape is entirely different. Modern cybercrime has evolved into a highly lucrative, heavily professionalized, and deeply organized global industry. Threat actors no longer operate in isolation; they function as sophisticated enterprises, complete with research and development divisions, customer service for ransomware victims, and affiliate programs.

More importantly, modern threat actors heavily leverage automation. Rather than manually targeting a single company, attackers utilize automated botnets, polymorphic malware, and artificial intelligence-driven scripts to scan millions of IP addresses across the globe in a matter of minutes. These automated tools tirelessly probe the internet for unpatched software vulnerabilities, weak or default passwords, and exposed administrative portals. Search engines like Shodan allow attackers to easily locate vulnerable internet-


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

connected devices worldwide. Once an automated script identifies a vulnerability, the exploitation phase—which can range from deploying ransomware to quietly establishing a backdoor for future data exfiltration—happens at machine speed, often long before a human IT administrator is even aware that a breach has occurred.

Historically, the cybersecurity industry has relied heavily on a "fortress mentality" to defend against these digital intrusions. Organizations built massive digital walls using static firewalls, signature-based Intrusion Detection Systems (IDS), Intrusion Prevention Systems (IPS), and endpoint antivirus software. The overarching philosophy of this era was strictly reactive: monitor the network perimeter, identify known malicious behavior based on a database of historical threat signatures, and block it from entering.

However, this reactive approach is fundamentally flawed and increasingly ineffective in the modern threat landscape. Traditional defenses are notoriously blind to zero-day exploits—vulnerabilities in software that are unknown to the vendor and have never been seen before, meaning no defensive signature exists to block them. Furthermore, Advanced Persistent Threats (APTs), which are often state-sponsored hacking groups, frequently bypass perimeter defenses entirely by utilizing legitimate, stolen user credentials. If an attacker logs into a network using a stolen, valid password, a traditional firewall will not stop them, because the traffic appears entirely legitimate.

Furthermore, as attackers increasingly utilize complex proxy chains, Virtual Private Networks (VPNs), and anonymizing networks like Tor to constantly rotate and hide their true IP addresses, simple blocklists (banning specific IP addresses) have become completely obsolete. The sheer volume of automated attacks bombarding network perimeters on a daily basis creates massive amounts of log data. This leads to a dangerous psychological and operational phenomenon known in the industry as "alert fatigue." Security Operations Center (SOC) analysts are routinely overwhelmed by thousands of low-level, often false-positive alerts every single shift. Sifting through this endless noise makes it incredibly difficult, if not impossible, to identify the genuine, critical threats that are quietly slipping through the cracks.

This growing asymmetry—where attackers leverage high-speed automation and vast resources, while defenders rely on manual review, reactive patching, and outdated signatures—has created an urgent mandate within the cybersecurity industry. There is a critical, unavoidable need to transition away from purely passive, reactive monitoring. The industry must move toward proactive, intelligence-driven defense strategies that can


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

anticipate attacks, deceive the attackers, and operate autonomously at the exact same machine speed as the threats they are designed to stop.

### 1.2 PARADIGM OF CYBER DECEPTION AND HONEYPOTS

To counter the inherent limitations and failures of traditional perimeter defenses, security researchers and enterprise defenders have increasingly turned to a strategy that is as old as warfare itself: deception. The concept of cyber deception fundamentally flips the traditional cybersecurity model on its head. Instead of solely focusing resources on keeping attackers out of the network—a task that is widely considered impossible in the long term—deception strategies operate on the pragmatic assumption that attackers are already at the gates, or have already quietly breached the outer perimeter. The goal of deception is not to build a taller wall, but to deliberately deploy fabricated vulnerabilities, decoy assets, simulated databases, and fake internal communications to attract, trap, confuse, and study the threat actors.

The foundational technology enabling this proactive defense strategy is the "Honeypot." Conceptually, a honeypot is an intentionally compromised network node, server, or application that is designed to look like a highly attractive target to an attacker. It appears to contain valuable proprietary information, vulnerable legacy services, or easily guessable administrative passwords. However, in reality, it is a tightly monitored, heavily instrumented trap. The core philosophy of a honeypot is based on its isolation: because a honeypot serves absolutely no legitimate business function and hosts no real production data, no authorized employee, customer, or automated internal service should ever attempt to interact with it. Therefore, any interaction whatsoever—whether it is a simple automated port scan, an attempted login, a brute-force dictionary attack, or a complex payload injection—is inherently classified as unauthorized, malicious, and worthy of immediate investigation. This virtually eliminates the "false positive" problem that plagues traditional IDS systems.

Historically, the academic and professional literature categorizes honeypots by their level of interaction. This classification dictates both the quality and depth of the forensic intelligence gathered, as well as the inherent risk associated with running the system on a corporate network:

**Low-Interaction Honeypots:** These are the most basic form of deception. These systems merely simulate the presence of basic network protocols and services. For example, a low-interaction honeypot might open Port 22 (SSH) or Port 21 (FTP) and



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

present a basic login banner. However, there is no real operating system behind the port. If an attacker attempts to log in, the honeypot simply records their IP address, the username and password they attempted to use, and the timestamp, and then drops the connection or returns a generic error. These systems are incredibly lightweight, require very few computing resources, and are highly secure because there is nothing for the attacker to actually compromise. However, their simplicity is also their downfall. Modern, sophisticated automated bots can easily "fingerprint" these systems. By analyzing the speed of the server's response or the exact phrasing of the error messages, attackers quickly realize they are interacting with a hollow simulation and immediately abandon the attack, yielding very little useful intelligence for the defender.

**High-Interaction Honeypots:** To counter the fingerprinting problem, researchers developed high-interaction honeypots. These systems involve deploying real, fully functional, and intentionally vulnerable operating systems and full-fledged applications. Because the environment is real, attackers can log in, download additional malware from the internet, install rootkits, and attempt to escalate their privileges. This allows defenders and malware researchers to capture incredibly rich, deep forensic data. They can study the complete lifecycle of an attack, reverse-engineer the specific malware payloads dropped by the attacker, and understand their specific tactics, techniques, and procedures (TTPs). However, high-interaction honeypots are highly resource-intensive to maintain and present a massive, terrifying security risk. If a high-interaction honeypot is not perfectly isolated from the rest of the corporate network (usually via strict VLANs and firewalls), a skilled attacker could compromise the honeypot and use it as a base of operations—a "pivot point"—to launch attacks against the actual, legitimate host network.

The HONEYTRAP project introduces a modernized, optimized hybrid approach, often referred to in modern literature as a medium-to-high fidelity application honeypot. Rather than hosting an entire vulnerable operating system (which is dangerous and heavy) or a simple open port (which is easily detected), this framework focuses on simulating specific, high-value web applications and protocol interfaces. By utilizing modern web development frameworks like Node.js and Express.js, the HONEYTRAP system can render highly realistic, functional clones of enterprise login portals—such as fake Amazon Web Services (AWS) management consoles, Microsoft 365 environments, or corporate VPN gateways.

This specific paradigm offers a distinct psychological and tactical advantage over the attacker. By presenting a highly convincing, visually accurate target, the honeypot forces the attacker to expend valuable time, computational resources, and effort attempting to


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

breach a completely worthless decoy. While the attacker believes they are successfully infiltrating a sensitive corporate database or executing a successful credential stuffing attack, the HONEYTRAP system is silently working in the background. It is extracting their true, unwrapped IP address, mapping their geospatial location via external APIs, and analyzing their behavioral metrics to determine their intent. This intelligence can then be fed back into the main corporate firewall to block the attacker network-wide. Ultimately, this approach transforms the attacker's own aggressive momentum into a defensive asset for the organization, turning the hunter into the hunted.

### 1.3 PROBLEM STATEMENT

Despite the proven theoretical value of cyber deception and the widespread academic praise for honeypot technologies, deploying traditional, open-source honeypots in a modern, fast-paced enterprise Security Operations Center (SOC) presents several critical operational flaws. Many legacy honeypots were designed as academic research tools, not as active enterprise defense mechanisms. The HONEYTRAP project was specifically conceived and engineered to directly address three major, systemic problems inherent in legacy deception architectures:

#### 1. Burden of Alert Fatigue and the Failure of Manual Mitigation

Traditional honeypots are entirely passive entities. They are exceptional at recording malicious activity, but they take absolutely no action to stop it. When a traditional honeypot is targeted by an automated brute-force botnet attack, it dutifully records every single password guess. This results in the generation of massive text files filled with hundreds of thousands of log entries. Human security administrators must then manually open these logs, parse through the data, identify the malicious IP addresses, and then log into their enterprise firewalls to manually write and apply block rules.

In the modern threat landscape, this manual intervention is fatal. In the five to ten minutes, it takes a human analyst to realize an attack is happening, parse the logs, and apply a block, an automated botnet could have already finished interacting with the honeypot, pivoted, and begun scanning and exploiting other genuinely vulnerable machines on the actual network. The reliance on human intervention creates an unacceptable bottleneck. The lack of automated, intelligent response mechanisms renders traditional honeypots too slow to mitigate modern, high-speed threats effectively.


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

### 2. The Absence of Zero-Latency Visualization and Context

During an active, ongoing cyberattack, immediate situational awareness is the most critical asset a security team can possess. Unfortunately, many existing forensic logging tools store their captured data in complex backend SQL databases or raw, unformatted flat text files (.txt or .json). To make sense of this data, security teams are forced to pipe the logs into expensive, complex third-party Security Information and Event Management (SIEM) software, such as Splunk, or the ELK stack (Elasticsearch, Logstash, Kibana).

These integrations are notoriously difficult to set up and frequently suffer from processing and indexing delays. Security teams lack a unified, standalone, out-of-the-box platform that provides real-time, zero-latency visualization of an ongoing attack. Without an immediate, graphical representation of where an attack is geographically originating from, what specific protocols are currently being targeted, and what credentials are being used, administrators cannot make rapid, informed decisions. Staring at scrolling lines of text in a terminal is not an effective way to manage a high-stress security incident.

### 3. Vulnerability of Forensic Evidence and the Chain of Custody

One of the most critical, yet frequently overlooked, vulnerabilities of standard digital forensics involve the preservation of the "Chain of Custody." When a security breach occurs, the resulting logs are not just operational data; they are potential legal evidence. They may be required for internal audits, compliance reporting, or formal criminal prosecution by law enforcement.

Traditional honeypots almost universally store their forensic logs locally on the host machine's physical hard drive. This creates a massive single point of failure. If an advanced human attacker (such as an APT) realizes they are trapped in a honeypot, their immediate priority shifts from data exfiltration to covering their tracks. They will often attempt to exploit the honeypot further to gain root access. Once they have administrative privileges, their first action is to execute commands to wipe the system's log directories (e.g., executing rm -rf /var/log/* in Linux environments). If the attacker successfully deletes, alters, or encrypts (via ransomware) these localized files, the forensic evidence is permanently destroyed. Because the data has been tampered with, it loses all investigative value and is rendered completely inadmissible in a court of law. There is an urgent need to separate the collection of forensic data from the localized storage of that data.



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

### 1.4 OBJECTIVES OF THE PROJECT

To systematically solve the severe operational and security flaws outlined in the problem statement, the primary objective of this project is to architect, develop, and implement the HONEYTRAP system. This system is envisioned not just as a passive logging tool, but as a proactive, AI-driven, and cryptographically secure cyber deception platform. To ensure a structured development lifecycle, the project’s specific goals are divided into two distinct, sequential phases of implementation.

**To design and deploy a High-Fidelity Deception Engine:** The first objective is to build a robust server architecture utilizing Node.js and Express.js that is capable of deploying multiple simultaneous, convincing traps. This includes creating visually accurate, interactive simulated web authentication portals hosted on HTTP (Port 8080). Furthermore, the system must deploy foundational protocol listeners for File Transfer Protocol (FTP on Port 2121), Secure Shell/Telnet (SSH on Ports 2222/2323), and simulated database connections (SQL on Port 3307). These traps must be designed to deeply engage attackers, tricking them into revealing their payload, without providing any real services that could be used as a pivot vector into the host network.

**To engineer a stealthy Forensic Extraction Module:** The system must accurately and silently intercept raw network socket connections. The logic must be capable of extracting the attacker's source IP address and stripping away any network routing anomalies (such as IPv6 wrapping or local proxy headers) to reveal the true origin. Once isolated, the system must asynchronously route this IP to an external Geolocation API to resolve the attacker’s precise physical coordinates (City, Country, ISP) without slowing down the main server's ability to absorb concurrent attacks.

**To construct a Zero-Latency Live Forensic Dashboard:** To solve the visualization problem, the objective is to build an interactive, graphical command center. By utilizing event-driven WebSocket architecture (Socket.io), the system must stream incoming threat intelligence directly from the backend server to the frontend user interface in real-time. This objective aims to achieve visualization latency of under 200 milliseconds, entirely bypassing the delays associated with traditional HTTP polling or batch log processing.

**To integrate an Artificial Intelligence (AI) Behavioral Microservice:** To solve the bottleneck of manual human mitigation, the project aims to integrate a separate Python-based Machine Learning microservice. This engine will be tasked with performing real-time behavioral analysis on incoming connections. By evaluating metadata such as


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

connection velocity, the timing between page load and form submission, and structural anomalies in the HTTP User-Agent headers, the AI must accurately classify the incoming threat as either a scripted automated "Botnet" or a manual "Human" operator.

**To implement an Autonomous Defense Mechanism (The "Kill Switch"):** Directly linked to the AI's classification, the system must be capable of defending itself. Upon receiving a high-confidence "Bot" classification from the Python microservice, the Node.js core server must autonomously trigger a network defense protocol. It must instantly terminate the malicious socket connection and automatically drop all future packets originating from the offending IP address. This effectively removes the human administrator from the immediate incident response loop, neutralizing threats at machine speed.

**To establish Secure Blockchain Persistence for Evidence:** To resolve the critical vulnerability of localized log tampering, the final objective is to integrate Web3 technology. Instead of writing the finalized forensic logs to a fragile local text file, the system will cryptographically hash the data using SHA-256 and execute a smart contract transaction to write this hash directly to a decentralized blockchain ledger. This guarantees the absolute immutability, permanence, and strict legal admissibility of the gathered evidence, even if the honeypot server itself is completely destroyed by an attacker.

## 1.5 SCOPE AND LIMITATIONS

Defining the exact operational boundaries of a cybersecurity tool is essential for its effective deployment and for managing the expectations of security personnel. The scope of the HONEYTRAP project encompasses the creation of an isolated, event-driven server environment that actively monitors specific, predefined network ports. The system is designed to simulate targeted enterprise web applications and essential network services, extract behavioral and geospatial metadata from incoming hostile traffic, classify the nature of the threat utilizing a supervised machine learning model, and permanently secure the resulting output on a decentralized blockchain ledger. The visualization scope is strictly limited to the data vectors displayed on the custom-built Admin Dashboard, primarily focusing on geographical attack origins, protocol distribution, AI verdicts, and captured raw credentials.

However, despite its advanced features, the system operates under certain architectural, theoretical, and environmental limitations that must be acknowledged:



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

- **Depth of Deception vs. Resource Constraints:** HONEYTRAP is a medium-to-high fidelity system; it is not a full-scale operating system emulator or a "digital twin" of a massive corporate network. While the simulated AWS or Microsoft 365 portals are highly convincing to automated botnets and casual human hackers, they do not possess deep backend functionality. If a highly skilled, persistent human penetration tester interacts with the system over an extended period, they will eventually notice the lack of complex database responses, realize they are in a simulated sandbox environment, and abandon the attack.
- **Machine Learning Model Drift and Evasion:** The accuracy of the AI-powered autonomous Kill Switch is inherently tied to the quality and relevance of its training data. The model heavily relies on interaction velocity (speed) to differentiate bots from humans. If advanced threat actors become aware of this behavioral analysis, they may re-engineer their botnets into "low-and-slow" attacks. By programming bots to deliberately introduce random delays, simulate typing errors, and mimic the erratic timing of a human user, attackers could potentially generate false negatives, tricking the AI into classifying the bot as a human, thereby bypassing the autonomous termination feature.
- **Blockchain Network Dependency and Latency:** Replacing traditional local file storage with decentralized blockchain logging introduces a heavy reliance on external network stability and consensus mechanisms. While writing data to a localized blockchain testnet (like Ganache) is instantaneous, migrating this feature to a public, mainnet blockchain (like Ethereum or Polygon) would introduce block confirmation delays. Furthermore, writing data to a public ledger incurs "gas fees" (transaction costs). During a high-volume Distributed Denial of Service (DDoS) attack, where the honeypot might log thousands of connections per second, writing every single event to a public blockchain could become financially prohibitive and computationally bottlenecked.
- **Deployment Isolation Requirements:** The HONEYTRAP system is explicitly designed as a parallel intelligence-gathering tool, not as an inline Intrusion Prevention System (IPS). It does not sit in front of actual production servers to filter their traffic; it sits alongside them. Therefore, it must be deployed in a strictly sandboxed environment or a separate Demilitarized Zone (VLAN/DMZ). If improperly configured on a production machine with excessive system privileges, the intentionally opened vulnerable ports could theoretically be leveraged by a


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

sophisticated attacker to exhaust the host machine's CPU or memory resources, inadvertently causing a denial of service to legitimate applications running on the same hardware.

### 1.6 ORGANIZATION

To provide a clear, logical, and academic progression of the research, structural development, and operational testing of the HONEYTRAP system, the remainder of this project report is rigorously structured into the following chapters:

- **Chapter 2: Literature Survey** provides a comprehensive review of the existing academic and industry research surrounding cyber deception. It traces the historical evolution of honeypots from simple open ports to modern emulators, examines the current state-of-the-art application of machine learning in network traffic analysis, and explores emerging studies on leveraging Web3 and blockchain technology for digital evidence preservation, culminating in a critical gap analysis that justifies the necessity of this project.
- **Chapter 3: Requirement Specification** outlines the foundational blueprint of the project. It clearly details the specific functional requirements (what the system must do, such as extracting IP headers and triggering the Kill Switch) and the non-functional requirements (how the system must perform, including strict latency constraints and security sandboxing protocols). It also defines the exact hardware, software, and external API prerequisites required for successful deployment.
- **Chapter 4: System Design** breaks down the architectural structure of the HONEYTRAP framework. It explains the multi-tiered Client-Server model in detail, mapping the complex data interactions between the Tier 1 Deception Layer, the Tier 2 Node.js/Python Core Logic processing hub, and the Tier 3 Visualization Dashboard. It includes explanations of the data flow sequences and the structural logic used to ensure the honeypot remains permanently isolated from the host network.
- **Chapter 5: Implementation and Results** transition the project from theoretical design to practical execution. It details the specific coding procedures, environments, and software libraries used to build the Node.js server, train the Python AI model, and deploy the Web3 smart contracts. It discusses the technical challenges faced during development and presents the empirical results of the system when subjected to simulated high-volume network attacks, followed by a


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

comparative analysis against existing traditional cybersecurity tools.

- **Chapter 6: Conclusion and Future Scope** summarize the project's overall technical achievements and its impact on the philosophy of proactive network defense. It reflects honestly on the current operational limitations of the system and outlines highly potential avenues for future research and development, including cloud-based Docker containerization, dynamic AI-generated deception assets, and integration with global threat intelligence feeds.


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

# CHAPTER 2

# LITERATURE SURVEY

## 2.1 INTRODUCTION

The study of cybersecurity is inherently an arms race between defenders and threat actors. As defenders build taller walls, attackers engineer longer ladders. To understand the current state-of-the-art in network defense, it is essential to review the historical literature and trace the evolution of the technologies that form the foundation of modern security practices. A comprehensive literature survey serves not merely as a summary of past works, but as a critical analysis of the successes, failures, and limitations of existing methodologies.

For decades, the predominant philosophy in network security literature was focused on perimeter defense and reactive monitoring. Early academic papers and industry standards heavily emphasized the deployment of robust firewalls, Intrusion Detection Systems (IDS), and signature-based antivirus software. The overarching goal was to create an impenetrable fortress. However, as the global digital infrastructure expanded into cloud computing, mobile networks, and the Internet of Things (IoT), the concept of a definable "perimeter" dissolved. Recent literature overwhelmingly agrees that a purely reactive defense posture is no longer sustainable. As threat actors have adopted automated botnets, polymorphic malware, and artificial intelligence, they have gained the ability to execute attacks at machine speed, easily bypassing static, signature-based defenses.

This realization has prompted a massive paradigm shift in both academic research and enterprise security strategies. The focus has pivoted toward "proactive defense" and "cyber deception." Instead of waiting for an attacker to breach a firewall, proactive defense involves actively luring, confusing, and studying the attacker within a controlled environment.

This chapter conducts an extensive review of the literature surrounding the four foundational pillars of the HONEYTRAP project. First, it examines the historical evolution of honeypot architectures, analyzing the transition from simple low-interaction decoys to high-fidelity application traps. Second, it reviews the literature on network visualization, focusing on the shift from delayed, flat-file logging to real-time, event-driven WebSocket architectures. Third, it explores the intersection of artificial intelligence and cybersecurity, specifically the use of machine learning for behavioral threat classification and automated mitigation. Finally, it investigates the emerging integration of decentralized blockchain


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

technology as a mechanism for guaranteeing the cryptographic integrity and legal admissibility of digital forensic evidence. By analyzing these domains, this survey identifies the critical gaps in current security frameworks that the HONEYTRAP system is designed to resolve.

## 2.2 EVOLUTION OF HONEYPOT ARCHITECTURES

The concept of cyber deception is not entirely new; its roots can be traced back to the late 1980s. The earliest documented instance of utilizing a decoy system to trap a hacker was detailed by Clifford Stoll in his seminal book The Cuckoo's *Egg*, where he famously created a fake network of documents to track an unauthorized user accessing the Lawrence Berkeley National Laboratory. Since then, the concept of the "honeypot"—a trap set to detect, deflect, or counteract attempts at unauthorized use of information systems—has become a heavily researched domain in computer science.

Over the years, the academic literature has categorized honeypots based on their "level of interaction." This metric defines how deeply an attacker is allowed to interact with the decoy system. The level of interaction directly correlates with both the quality of the forensic intelligence gathered and the level of risk the system poses to the host network.

### 2.2.1 LOW-INTERACTION HONEYPOTS

Early research in the field of cyber deception focused heavily on low-interaction honeypots. These systems were designed to be incredibly lightweight, easily deployable, and highly secure. A low-interaction honeypot does not provide a real, functional operating system for the attacker to interact with. Instead, it utilizes software to merely simulate the presence of basic network protocols and listening ports.

For example, a low-interaction honeypot might be configured to open Port 21 to simulate a File Transfer Protocol (FTP) server, or Port 22 to simulate a Secure Shell (SSH) login. When a threat actor (or an automated scanning bot) connects to the port, the honeypot presents a static, pre-programmed "banner" (the text that identifies the server type). If the attacker attempts to log in, the honeypot simply records the attacker's IP address, the timestamp, and the attempted username and password. Once the data is logged, the honeypot typically drops the connection or returns a generic "Authentication Failed" error message. Famous early examples in the literature include Honeyd, a daemon that could simulate entire networks of virtual hosts, and Dionaea, which was designed specifically to capture malware by simulating vulnerable Windows services.

The primary advantage highlighted in the literature for low-interaction honeypots is safety. Because there is no real operating system, there is no way for an attacker to break out of the honeypot and use it to attack the rest of the corporate network. Furthermore,


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

because they are merely simple scripts listening on a port, they consume virtually no CPU or memory resources, allowing defenders to deploy thousands of them across a network.

However, the literature is equally clear on their massive drawbacks. Modern cyber-attacks are largely automated. Attackers utilize sophisticated tools like Nmap (Network Mapper) or specialized vulnerability scanners. These tools are designed to "fingerprint" a server. They analyze the exact timing of the TCP handshake, the structure of the packet headers, and the specific phrasing of error messages. Because low-interaction honeypots are just simple simulations, automated bots can easily fingerprint them. The bot quickly realizes it is talking to a dummy script rather than a real server, aborts the attack, and moves on. Consequently, low-interaction honeypots fail to gather deep intelligence; they can log an IP address, but they cannot capture the advanced malware payloads or complex exploitation techniques used by modern threat actors.

### 2.2.2 HIGH-INTERACTION HONEYPOTS

To solve the fingerprinting problem and gather deeper forensic intelligence, researchers and security professionals pivoted toward high-interaction honeypots. The literature defines a high-interaction honeypot as a fully functional, real operating system (such as a full installation of Ubuntu Linux or Windows Server) that is intentionally deployed with known vulnerabilities, weak passwords, or unpatched software.

Because high-interaction honeypots are actual systems, there is nothing for an automated scanner to "fingerprint" as fake. When an attacker breaches a high-interaction honeypot, they are granted a real command-line shell. They can navigate the file system, download additional malicious tools from the internet, install rootkits, and attempt to establish persistence. This level of interaction is a goldmine for cybersecurity researchers. By silently monitoring the attacker's activities at the kernel or hypervisor level, defenders can capture zero-day malware, study the attacker's exact keystrokes, and understand their multi-stage tactics, techniques, and procedures (TTPs).

Despite their immense value in threat intelligence gathering, the academic consensus highlights severe operational challenges that prevent high-interaction honeypots from being widely deployed in standard enterprise environments. The most glaring issue is the immense security risk, often referred to in the literature as the "liability factor." Because the attacker is given access to a real, functional machine, there is a constant, terrifying risk that a highly skilled attacker could disable the monitoring software, break out of the honeypot's network isolation, and use the compromised machine as a "pivot point" to launch devastating attacks against the actual corporate production network.

Furthermore, high-interaction honeypots are incredibly expensive to maintain. They


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

require dedicated hardware or heavy Virtual Machines (VMs), complex hypervisor configurations, and constant supervision by highly trained security analysts to ensure they have not been completely compromised. Because of these factors, high-interaction honeypots are generally restricted to academic research labs and specialized malware analysis organizations, rather than standard enterprise defense.

### 2.2.3 MODERN HYBRID APPROACH

Recognizing that low-interaction honeypots are too simple and high-interaction honeypots are too dangerous, recent cybersecurity literature has focused on developing a middle ground. This is broadly categorized as the medium-to-high fidelity application honeypot.

Instead of emulating just an open port or providing an entire vulnerable operating system, the modern hybrid approach focuses on perfectly simulating specific, high-value applications. With the advent of containerization (like Docker) and modern web server frameworks (like Node.js and Express.js), researchers found they could build highly interactive, incredibly convincing web applications that trap attackers without exposing the underlying host operating system.

The HONEYTRAP project is built precisely upon this modern literature. Instead of running a heavy Windows Server VM, the system utilizes Node.js to host perfect visual and functional clones of highly targeted enterprise assets, such as Amazon Web Services (AWS) login portals, Microsoft 365 sign-in pages, and SQL database administration panels.

To an automated bot or a human hacker, the web portal looks and behaves exactly like the real thing. They can interact with the form fields, submit credentials, and receive convincing (but artificial) loading delays and error messages. This deep psychological deception forces the attacker to stay engaged and execute their payloads. However, because the backend is merely a Node.js event loop specifically programmed to extract data and drop the connection into a void, there is absolutely no pathway for the attacker to gain a real command shell. This modern approach successfully captures the rich, actionable intelligence of a high-interaction system while maintaining the strict, sandboxed safety of a low-interaction system.

## 2.3 REAL-TIME THREAT VISUALIZATION AND EVENT-DRIVEN ARCHITECTURE

A significant gap identified in legacy honeypot research and traditional network monitoring is the profound delay in threat visualization and incident awareness. Historically, network security relied almost entirely on static logging mechanisms. When


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

a firewall or a traditional honeypot intercepted an attack, it generated a line of text and appended it to a localized flat file (such as a .log or .txt file) or sent it via the Syslog protocol to a central storage server.

The literature points out that this batch-processing approach is fundamentally incompatible with the speed of modern cyber warfare. In older setups, a human security administrator would have to manually open these text files, write complex regular expression (Regex) queries to parse the data, and attempt to manually correlate an IP address to an attack vector. Even with the introduction of expensive Security Information and Event Management (SIEM) platforms, such as Splunk or the ELK stack, there remained a noticeable processing delay. Data had to be ingested, indexed, and queried before it could be rendered on a dashboard. By the time an analyst saw the alert on their screen, the automated attack had often already concluded, and the botnet had moved on.

Current literature in web-based network monitoring emphasizes the absolute necessity of real-time, zero-latency situational awareness. To achieve this, researchers have explored the integration of event-driven software architectures into cybersecurity tools. Node.js, built on Google Chrome's V8 JavaScript engine, is highly praised in recent computer science studies for its non-blocking, asynchronous Input/Output (I/O) model. Unlike traditional threaded web servers (like Apache) which create a new, memory-heavy thread for every single connection, Node.js uses a single-threaded event loop. This makes it exceptionally efficient at handling thousands of rapid, concurrent network connections—such as those experienced during a Distributed Denial of Service (DDoS) or botnet brute-force attack—without crashing or bottlenecking.

Furthermore, the integration of WebSocket technology into security dashboards has revolutionized threat visualization. Traditional web dashboards rely on the HTTP protocol, which operates on a request-response model. The dashboard must constantly "poll" or ask the server, "Is there new data?" every few seconds. This creates unnecessary network traffic and inherent delays. WebSocket's (such as the Socket.io library used in the HONEYTRAP system), conversely, establish a persistent, full-duplex, open connection between the server and the browser.

The literature shows that utilizing WebSocket's allows forensic data to be pushed actively from the honeypot's core logic directly to the security dashboard the exact millisecond the data is extracted. This event-driven architecture successfully bypasses the indexing delays of traditional SIEMs, allowing the system to plot an attacker's geolocation on an interactive global map with a latency of under 150 milliseconds. This grants human operators an unprecedented, zero-latency view of the attack landscape as it unfolds in real


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

time.

## 2.4 ARTIFICIAL INTELLIGENCE IN BEHAVIORAL THREAT CLASSIFICATION

As the sheer volume and velocity of cyberattacks have scaled exponentially, the cybersecurity industry has reached a critical bottleneck: the human analyst. Security teams simply cannot manually review every captured IP address, analyze the payload, and execute firewall block rules fast enough to stop automated botnets. Consequently, the intersection of Artificial Intelligence (AI), specifically Machine Learning (ML), and network security has become one of the most heavily researched and funded topics in modern computer science literature.

Earlier literature on network defense focused on static blacklisting and signature matching. If an IP address was known to be bad, it was added to a list and blocked. If a payload matched the exact hash of a known virus, it was stopped. However, modern threat actors utilize proxy chains, VPNs, and botnets to constantly rotate their IP addresses, rendering static blacklists useless. Furthermore, attackers utilize polymorphic malware that changes its signature with every deployment.

To combat this, modern research focuses on behavioral and heuristic analysis using Artificial Intelligence. Instead of asking, "Have I seen this IP address before?" machine learning models are trained to ask, "Does the behavior of this connection look like a human being, or an automated script?"

In academic testing, supervised machine learning models—such as Random Forests, Support Vector Machines (SVM), and Decision Trees—have proven highly effective at this type of classification. These models are trained on massive datasets of network traffic metadata. When applied to a honeypot environment like HONEYTRAP, the AI evaluates several critical behavioral metrics:

- **Interaction Velocity:** An automated script can complete a TCP handshake, parse an HTML page, fill out a login form, and submit a payload in a few milliseconds. A human user requires several seconds to visually process the page, move the mouse, type the password, and click submit.
- **Header Anomalies:** Bots frequently utilize randomized or stripped User-Agent headers that do not match standard browser behaviors (e.g., claiming to be a Chrome browser but lacking the standard accepted language headers).
- **Request Intervals:** Botnets often attack at perfectly uniform intervals (e.g., exactly one guess every 0.5 seconds), whereas human interaction is highly erratic.

The literature highlights that by integrating an AI classifier directly into the honeypot's


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

core logic, systems can transition from merely *detecting* threats to *autonomously responding* to them. This represents a monumental shift in active defense. When the Python AI microservice in the HONEYTRAP architecture classifies a connection as a "Bot" with high confidence, it can instantly trigger a "Kill Switch." This instructs the server to drop the socket connection and block the IP at the network level in milliseconds, entirely removing the human bottleneck in incident response and stopping the attack dead in its tracks.

### 2.5 BLOCKCHAIN TECHNOLOGY FOR FORENSIC DATA INTEGRITY

One of the most persistent and critical challenges discussed in digital forensics and cyber law literature is the preservation of the "Chain of Custody." When a security event occurs, the resulting digital logs are used as evidence to understand how the breach happened, to satisfy regulatory compliance audits, and, in severe cases, for legal prosecution in a court of law.

However, digital evidence is inherently fragile. Traditional honeypots and servers store their forensic logs locally on their physical hard drives. If an Advanced Persistent Threat (APT) or a highly skilled human attacker manages to escalate their privileges and breach the host system, their very first priority is almost always to cover their tracks. They achieve this by altering, encrypting (via ransomware), or completely deleting the local forensic logs (e.g., executing commands to wipe the /var/log directory).

The legal and academic literature dictates that if digital logs are tampered with, or if the defense cannot prove they *haven't* been tampered with, the logs lose all court admissibility and investigative value.

To solve this vulnerability, recent cybersecurity research has increasingly turned to Web3 and Blockchain technologies. A blockchain is a decentralized, distributed, digital ledger. Its core mathematical properties make it cryptographically secure and, most importantly, "append-only." Once data is written to a block and mathematically verified by the network consensus, it becomes practically immutable. It cannot be altered, overwritten, or deleted by any single party without altering all subsequent blocks and alerting the entire network.

Recent studies have proposed integrating blockchain directly with IoT security systems and network logging mechanisms to create tamper-proof forensic vaults. By taking the final JSON threat log generated by a honeypot, generating a unique SHA-256 cryptographic hash of that data, and writing that hash to a smart contract on a decentralized ledger (such as Ethereum), security professionals can guarantee the absolute integrity of their evidence.


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

The HONEYTRAP framework builds directly upon this cutting-edge area of study. By implementing a Secure Blockchain Persistence module, the system replaces vulnerable local file storage with a decentralized vault. Even in a worst-case scenario where the physical honeypot server is completely compromised, wiped clean, or physically destroyed by an attacker, the hashed forensic evidence survives completely intact on the blockchain. Any investigator can take the remaining data, hash it, and compare it to the blockchain record to prove mathematically that the evidence has not been altered since the millisecond the attack occurred.

## 2.6 SUMMARY OF LITERATURE AND GAP ANALYSIS

An extensive and critical review of the existing academic literature reveals a rapidly maturing field of proactive cyber defense. Honeypots have evolved from simple academic curiosities into vital enterprise tools. Event-driven web architectures have enabled real-time data processing. Artificial Intelligence has proven capable of classifying high-speed threats. Blockchain technology has offered a mathematical solution to evidence tampering. However, the literature review also reveals a glaring, distinct gap: fragmentation.

While all of these technologies are thoroughly researched and proven in isolation, there is a distinct lack of unified, holistic frameworks that combine these elements into a single, cohesive, out-of-the-box system.

- Many proposed AI-powered honeypots still rely on vulnerable, centralized SQL databases for storage.
- Conversely, early prototypes of blockchain-logging honeypots are often entirely passive; they secure the data brilliantly, but lack real-time graphical visualization or any autonomous defensive capabilities.
- Traditional open-source tools (like Cowrie or Dionaea) lack both AI integration and blockchain persistence, requiring organizations to piece together expensive, complex SIEM integrations just to make the data readable.

The HONEYTRAP project was engineered specifically to address and bridge this critical research gap. By synthesizing high-fidelity Node.js application deception, zero-latency WebSocket visualization, autonomous Machine Learning behavioral analysis, and immutable decentralized Blockchain evidence preservation, this project proposes a completely unified, end-to-end proactive defense framework. It moves beyond theoretical isolation to provide a highly relevant, scalable, and intelligent solution tailored to the rigorous demands of modern enterprise security operations and digital forensic investigations.


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

# CHAPTER 3

# REQUIREMENT SPECIFICATION

## 3.1 INTRODUCTION

The Requirement Specification chapter serves as the foundational architectural blueprint for the entire HONEYTRAP project. In standard software engineering, a requirement specification outlines how a user interacts with a system to achieve a specific business goal. However, designing a proactive cybersecurity and cyber-deception platform requires a radically different approach. The HONEYTRAP system is not a standard application designed for ease of use by a traditional consumer; it is a weaponized decoy system designed to operate in a hostile environment, attract malicious actors, extract intelligence, and autonomously defend the network.

Because the system is designed to deliberately invite cyberattacks, the requirements outlined in this chapter are governed by a strict philosophy of isolation, speed, and automation. If a standard web application fails, a user receives a 404 error. If a honeypot fails or is improperly specified, it becomes a pivot point for a hacker to destroy the host network. Therefore, the specifications detailed here focus heavily on high-speed data processing, seamless microservice integration, strict sandboxing, and the preservation of forensic integrity under duress.

This chapter comprehensively defines the specific functional capabilities the system must possess (Functional Requirements), the qualitative standards and security constraints it must adhere to (Non-Functional Requirements), and the foundational environments necessary to run the code (System Requirements). The criteria documented here act as the definitive checklist for the successful implementation and validation of both the Phase I (Deception) and Phase II (Automation and Blockchain) objectives.

## 3.2 OVERALL SYSTEM DESCRIPTION

### 3.2.1 PRODUCT PERSPECTIVE

The HONEYTRAP framework operates as a standalone, decentralized, and entirely autonomous defense mechanism. It is critical to define what the system is *not*: it is not an inline firewall, and it is not an administrative portal or standard Intrusion Prevention System (IPS) that sits directly in front of legitimate production servers to filter their traffic.

Instead, the product operates in parallel to the actual network. It is strategically



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

deployed within a network's Demilitarized Zone (DMZ) or placed in a highly isolated Virtual Local Area Network (VLAN). The product is architected as an event-driven ecosystem consisting of specific processing tiers:

1. **The Deception Edge:** This is the outermost layer, directly exposed to the internet. It broadcasts fake services and vulnerable web interfaces to absorb incoming attacks.
2. **The Core Processing Hub:** Powered by Node.js and a Python artificial intelligence microservice, this tier silently processes the attacks, extracts intelligence, and autonomously makes blocking decisions.
3. **The Immutable Ledger:** Utilizing Web3 integration, this component securely stores the forensic data on a decentralized blockchain network.
4. **The Passive Visualizer:** A strictly read-only, localized visualization feed. To drastically reduce the system's own attack surface, the architecture explicitly omits any web-based administrative portal. There is no central login page for human defenders to manage the system remotely, ensuring that attackers cannot compromise the honeypot's management plane.

### 3.2.2 USER CLASSES AND CHARACTERISTICS

Traditional software defines users by their roles and permissions. In the context of the HONEYTRAP system, the "users" are divided into entirely distinct, opposing classes with completely different characteristics and intents.

- **The Threat Actors (The Targets):** This class comprises the hostile entities interacting with the exposed edge of the system. This group is further subdivided into two categories:
  - Automated Botnets: Scripts running at machine speed, executing thousands of dictionary attacks, port scans, and credential stuffing attempts per minute. They are characterized by highly uniform interaction speeds and rigid programmatic logic.
  - Human Hackers: Live operators manually probing the system. They are characterized by erratic timing, manual exploration of the fake web forms, and targeted payload injections. The system must convincingly deceive both subgroups.


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

- **The Autonomous AI (The Defender):** Because the system omits a manual administrative portal, the Python machine learning microservice effectively acts as the primary "administrative user." It is responsible for analyzing the behavioral data in real-time, classifying the threat actor, and automatically enforcing network-level bans without requiring human input.
- **The Security Observers (The Audience):** Human cybersecurity analysts and forensic investigators interact with the system strictly as passive observers. They utilize the local visualizer to monitor real-time threat maps and export the cryptographically verified blockchain logs for post-incident legal analysis. They do not actively manage the traps during an ongoing attack.

### 3.2.3 OPERATING ENVIRONMENT

The HONEYTRAP system is designed to run in a cross-platform environment, though a Unix-based architecture is strictly preferred for optimal network-level control. The operational environment must support the binding of low-level network ports (which typically requires specific elevated permissions) while maintaining a strict sandbox around the application execution layer.

The environment requires completely unrestricted outbound internet access to communicate with external APIs (for geographical IP resolution) and decentralized blockchain nodes (for writing smart contracts). The environment must also support a localized, high-speed inter-process communication (IPC) bridge to allow the Node.js server to instantly pass extracted connection metrics to the Python AI microservice.

## 3.3 FUNCTIONAL REQUIREMENTS

Functional requirements explicitly define the precise behaviors, logic sequences, and data processing tasks the system must execute when subjected to specific conditions (i.e., when an attack occurs).

### FR-01: High-Fidelity Web Trap Deployment

- **Description:** Upon initialization, the system must utilize Express.js middleware to automatically deploy and host simulated, high-fidelity web applications on HTTP Port 8080.
- **Specifics:** The system must render visually accurate clones of targeted enterprise platforms, specifically an Amazon Web Services (AWS) login portal and a Microsoft 365 sign-in interface.


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

- **Processing Logic:** The routes must be programmed to accept HTTP POST requests. They must simulate artificial processing latency (e.g., a 1.5-second delay) before returning a fabricated "Authentication Failed" error, tricking automated tools into believing they are interacting with a real, slow corporate database.

**FR-02: Multi-Protocol Decoy Listeners**

- **Description:** To capture a wider spectrum of cyber threats beyond web application attacks, the system must deploy simulated protocol listeners on commonly targeted network ports.
- **Specifics:** The system must actively listen on Port 2121 (simulating FTP), Ports 2222/2323 (simulating SSH and Telnet), and Port 3307 (simulating an SQL database connection).
- **Processing Logic:** These listeners must accept incoming TCP handshakes, display a convincing service banner, and keep the connection alive just long enough to extract the initial payload before silently terminating the interaction.

**FR-03: Real-Time Payload and Socket Interception**

- **Description:** The core Node.js logic must intercept and extract identifying data from every unauthorized connection made to the traps defined in FR-01 and FR-02.
- **Specifics:** The system must access the raw socket connection to extract the incoming source IP address. It must utilize regular expressions to strip away any IPv6 wrapping (e.g., converting ::ffff:192.168.1.1 to a clean IPv4 address). It must also extract the User-Agent header, the exact timestamp of the interaction, and any credentials (usernames/passwords) submitted to the web traps.

**FR-04: Asynchronous IP Geolocation Resolution**

- **Description:** The system must translate the raw IP address captured in FR-03 into actionable geospatial intelligence.
- **Specifics:** The Node.js server must execute an outbound, non-blocking HTTP request to an external Geolocation API (e.g., IP-API).
- **Processing Logic:** The system must parse the returning JSON payload to extract the attacker's physical location, specifically targeting the City, Country, Latitude, Longitude, and Internet Service Provider (ISP). Crucially, this function must run


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

asynchronously to prevent blocking the main server thread from accepting concurrent attacks.

**FR-05: AI-Driven Behavioral Analysis**

- **Description:** The system must route the extracted connection metadata to the Python Artificial Intelligence microservice for real-time threat classification.
- **Specifics:** The AI model must ingest heuristic data points, including the time elapsed between the initial connection and payload submission, request intervals, and HTTP header anomalies.
- **Processing Logic:** Using a supervised machine learning algorithm (such as a Random Forest classifier), the Python microservice must calculate a probability score and return a binary classification to the main server: either "Bot" (automated script) or "Human" (manual operator).

**FR-06: Autonomous Network Termination (The Kill Switch)**

- **Description:** Based entirely on the verdict generated in FR-05, the system must execute an automated defensive response.
- **Specifics:** If the AI classifier returns a "Bot" verdict, the system must autonomously execute a network defense protocol to terminate the threat.
- **Processing Logic:** The system will immediately drop the active socket connection and blacklist the offending IP address in the local routing table to ignore all subsequent packets. This requirement must be executed entirely autonomously by the core logic; there is no administrative portal, and therefore no manual human override is required or permitted for this action.

**FR-07: Standardization and Cryptographic Hashing**

- **Description:** The system must compile all gathered intelligence into a standardized format and mathematically secure it.
- **Specifics:** Data from FR-03 (Payloads), FR-04 (Geolocation), and FR-05 (AI Verdict) must be merged into a single, strictly formatted JSON Threat Object.
- **Processing Logic:** The system must then pass this JSON object through a SHA-256 cryptographic hashing algorithm, generating a unique, fixed-length alphanumeric string that represents the exact state of the forensic data at that specific millisecond.


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

**FR-08: Smart Contract Execution and Blockchain Persistence**

- **Description:** To guarantee the preservation of the chain of custody, the system must store the cryptographic proof on a decentralized ledger.
- **Specifics:** Utilizing Web3 libraries, the Node.js server must connect to a blockchain network (e.g., a local Ethereum testnet like Ganache).
- **Processing Logic:** The system must execute a smart contract transaction, writing the SHA-256 hash generated in FR-07 directly onto the blockchain. This action replaces vulnerable local file storage, ensuring the data is immutable and legally admissible.

**FR-09: WebSocket Data Streaming**

- **Description:** The system must provide real-time situational awareness to local security observers without relying on standard web page reloads.
- **Specifics:** The system must establish a persistent WebSocket (Socket.io) connection between the server core and the local passive visualizer interface.
- **Processing Logic:** The instant the JSON Threat Object is finalized, the server must emit the data payload through the open WebSocket channel, dynamically updating the observer's visual graphs and threat maps instantly.

## 3.4 NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements dictate the qualitative standards, performance constraints, and strict security parameters that the system must meet. These define *how well* the system performs its functions, which is critical in a high-stress cybersecurity context.

**NFR-01: Performance and Zero-Latency Visualization**

- **Constraint:** The system must process incoming threats with extreme efficiency. The total round-trip time—from the exact moment a threat actor submits a payload, through geolocation API resolution, AI classification, blockchain hashing, and final emission to the passive visualizer—must average less than 150 milliseconds.
- **Rationale:** In a live cyberattack, seconds matter. Any delay in visualization or automated response could allow a botnet time to scan and exploit actual production systems.

**NFR-02: System Resource Efficiency and Concurrency**


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

- **Constraint:** The core server architecture must be capable of handling massive spikes in malicious traffic, such as a Distributed Denial of Service (DDoS) style credential stuffing attack, without crashing.
- **Rationale:** The system utilizes Node.js specifically for its non-blocking event loop. The system must process hundreds of concurrent connections simultaneously without exhausting the host machine's CPU or RAM, maintaining a lightweight operational footprint even under heavy duress.

**NFR-03: Absolute Sandboxing and Isolation (The No-Pivot Rule)**

- **Constraint:** The honeypot must be fundamentally incapable of being used as a staging ground for further attacks.
- **Rationale:** The mock web portals and protocol listeners must act as "black holes." They do not connect to any real backend databases, and they must execute with the absolute minimum system privileges necessary to bind network ports. There must be zero exploitable pathways, reverse shell capabilities, or lateral movement vectors available to an attacker, even if they realize they are in a deceptive environment.

**NFR-04: Elimination of the Administrative Attack Surface**

- **Constraint:** The system must not host, expose, or contain any form of web-based administrative portal, login screen, or remote management interface.
- **Rationale:** Administrative portals are highly prized targets for threat actors. By completely removing the admin portal and relying entirely on the autonomous AI for mitigation and a local, read-only passive interface for visualization, the system dramatically reduces its own attack surface and eliminates the risk of compromised management credentials.

**NFR-05: Forensic Data Immutability and Legal Admissibility**

- **Constraint:** The system must guarantee that any forensic log successfully captured cannot be altered, encrypted, or deleted by any party, including the system's own creators.
- **Rationale:** If a honeypot is compromised, an attacker will attempt to wipe the logs to cover their tracks. By enforcing mandatory blockchain persistence, the system ensures that the historical evidence remains cryptographically intact on the decentralized ledger, preserving the strict chain of custody required for legal


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

prosecution and forensic audits.

**NFR-06: Fault Tolerance and API Fallbacks**

- **Constraint:** The system must handle failures in external dependencies gracefully.
- **Rationale:** If the external Geolocation API experiences an outage, or if the blockchain network suffers severe congestion and cannot accept new smart contracts instantly, the main Node.js event loop must not crash. The system must implement robust try/catch error handling, logging the raw IP locally as a temporary fallback while continuing to intercept new incoming attacks without interruption.

## 3.5 SYSTEM REQUIREMENTS

To fulfill the rigorous functional and non-functional specifications detailed above, the deployment environment must meet the following precise hardware and software prerequisites.

### 3.5.1 SOFTWARE REQUIREMENTS

The HONEYTRAP framework relies on a modern, deeply integrated stack of open-source frameworks and libraries.

- **Core Runtime Environment (Backend):** Node.js (Version 18.x LTS or higher) is required to power the high-speed, non-blocking asynchronous event loop.
- **AI Microservice Environment:** Python (Version 3.10 or higher) is required to host the machine learning models, ensuring compatibility with modern data science libraries.
- **Primary Frameworks & Libraries:** * Express.js: Required for routing and serving the deceptive high-fidelity HTTP web portals.
    - Socket.io: Required to maintain the persistent WebSocket connections for zero-latency data streaming to the passive visualizer.
    - Scikit-learn and Pandas: Required within the Python environment for structuring heuristic data and executing the Random Forest classification algorithms.
    - Web3.js: Required to bridge the Node.js backend with the decentralized blockchain network.
- **Blockchain Network:** A local Ethereum development environment, such as


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

Ganache, is required for localized testing and smart contract deployment without incurring public mainnet transaction fees.

- **External API:** A reliable, high-speed Geolocation REST API (such as IP-API or IPinfo) that supports high-volume, rapid concurrent asynchronous JSON requests.
- **Operating System:** A Unix-based distribution, such as Ubuntu Server 22.04 LTS, is highly recommended for its native networking capabilities, robust security permissions, and stability in hosting Node/Python dual environments.

### 3.5.2 HARDWARE REQUIREMENTS

Because the system is designed to host simulated environments, process complex machine learning algorithms, and interact with cryptographic ledgers simultaneously, the hardware must be robust enough to prevent internal bottlenecks during an attack.

- **Processor (CPU):** A modern multi-core processor (e.g., Intel Core i5/i7 10th Gen, AMD Ryzen 5, or equivalent server-grade CPU). High multi-threading capabilities are strictly required to ensure that the heavy computational tasks (like the AI classifier and SHA-256 blockchain hashing) run on separate threads or processes and do not block the main Node.js network interception thread.
- **Memory (RAM):** A minimum of 16 GB of DDR4 RAM is required. The system must simultaneously run the deceptive web servers, load the trained Python machine learning models directly into memory for instant classification, and maintain the local blockchain node's state, all of which require significant memory overhead.
- **Storage:** A minimum of 50 GB of available Solid State Drive (SSD) storage. High-speed read/write operations (IOPS) are critical for rapid ledger synchronization, smart contract execution, and the temporary caching of incoming high-volume threat data before it is hashed.
- **Network Interface:** A highly stable, high-bandwidth Network Interface Card (NIC) capable of handling gigabit speeds. The system requires rapid inbound throughput to absorb botnet attacks and equally stable outbound connectivity to communicate seamlessly with external APIs and global decentralized nodes without latency spikes.


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

# CHAPTER 4

# DESIGN

## 4.1 INTRODUCTION

The System Design chapter represents the critical bridge between the theoretical requirements defined in Chapter 3 and the practical software implementation detailed in Chapter 5. While the Requirement Specification outlined exactly *what* the system must accomplish, this chapter comprehensively defines *how* the system will achieve those objectives. Designing a proactive cybersecurity deception platform is fundamentally different from designing standard enterprise software. In standard software architecture, the primary goals are usability, accessibility, and seamless data sharing. Conversely, designing a honeypot requires engineering a highly attractive, weaponized decoy that must remain perfectly isolated to prevent an attacker from compromising the underlying host machine.

The architectural blueprint of the HONEYTRAP framework is built upon a philosophy of modularity, speed, and strict compartmentalization. The system must appear extremely vulnerable to external threat actors to attract engagement, yet internally, it must be impenetrable. To achieve this delicate balance, the design utilizes a specialized multi-tier architecture, completely separating the outward-facing deceptive elements from the internal processing logic and the secure data storage mechanisms.

This chapter provides a deep, granular analysis of the system's architecture. It dissects the three primary tiers of the Client-Server model, detailing the specific technologies and structural decisions that power the deception layer, the core logic hub, and the local visualization interface. Furthermore, this chapter outlines the specific module designs—including the high-fidelity web traps, the artificial intelligence microservice, and the blockchain persistence mechanism. By mapping out the precise data flow sequences, defining the logical classes, and establishing rigorous security isolation protocols, this design chapter serves as the definitive structural roadmap for the entire project.

## 4.2 OVERALL SYSTEM ARCHITECTURE

Modern software engineering heavily favors decoupled, service-oriented architectures, and cybersecurity tools are no exception. A monolithic design—where the user interface, backend logic, and database are intertwined into a single application—is highly dangerous for a honeypot. If an attacker finds a vulnerability in a monolithic honeypot, they gain access to the entire system. To mitigate this risk, the HONEYTRAP framework employs a strict, modern Three-Tier Architecture. This separation of concerns ensures that even if the


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

outer deception layer is completely overwhelmed by a massive botnet attack, the core logic and forensic storage tiers remain completely insulated and operational.

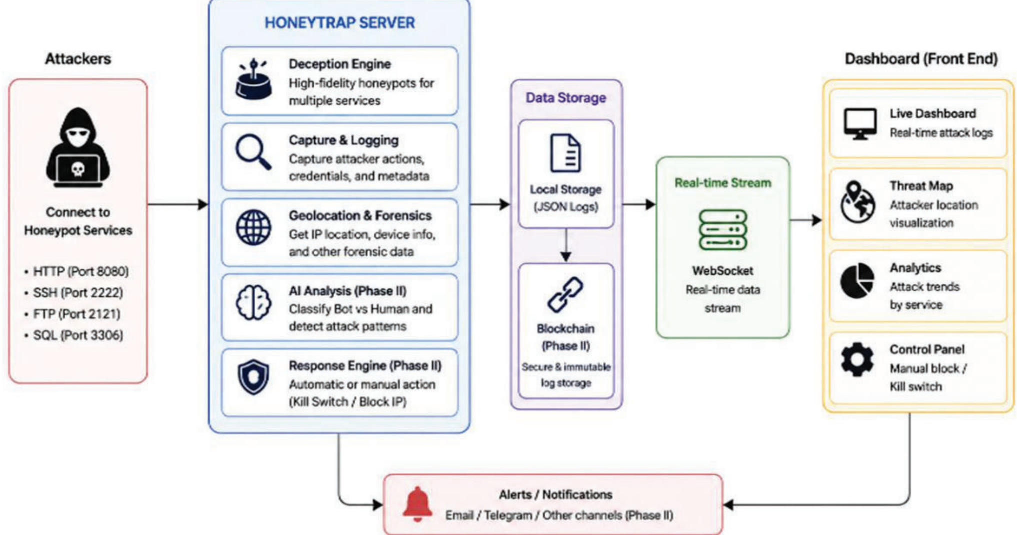


**Fig 4.1 System Architecture**

### 4.2.1 TIER 1: THE DECEPTION LAYER

Tier 1 is the outermost edge of the system's architecture. This is the only component of the system that is directly exposed to the internet or the internal network's Demilitarized Zone (DMZ). Its sole architectural purpose is psychological deception and traffic absorption.

From a design perspective, this layer consists of multiple independent, lightweight listening services bound to commonly targeted network ports. Instead of exposing real corporate services, this tier runs simulated endpoints designed to lure attackers. The architecture dictates that this layer must be incredibly lightweight. It does not perform any complex calculations, AI classifications, or database lookups. It acts purely as a "sponge" for malicious traffic.

The deception layer is structurally divided into two sub-categories:

- **Application-Level Traps (HTTP):** Hosted via Express.js on Port 8080 (simulating standard web traffic), this tier serves highly convincing, interactive HTML/CSS clones of popular enterprise login portals. The design ensures that these front-end interfaces are visually indistinguishable from legitimate services.
- **Protocol-Level Traps (TCP):** To capture automated scanners that ignore web traffic, this layer also opens specific TCP sockets on Port 2121 (FTP), Port 2222 (SSH), and Port 3307 (SQL). The design of these listeners is purely superficial; they complete the initial TCP three-way handshake and display a fake service


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

banner to trick the attacker into sending their first payload, but they possess no actual backend functionality.

#### 4.2.2 TIER 2: CORE LOGIC AND PROCESSING

Tier 2 is the centralized brain of the HONEYTRAP system. Unlike Tier 1, which faces the hostile internet, Tier 2 operates entirely behind the scenes in a secure processing environment. The primary design challenge for this tier is concurrency: it must be capable of processing hundreds of simultaneous attacks without dropping connections or slowing down.

To achieve this, the architecture utilizes Node.js as the primary backend engine. The design leverages the Node.js single-threaded, non-blocking asynchronous event loop. When Tier 1 intercepts an attack, it instantly passes the raw socket data to Tier 2. Because Node.js handles I/O operations asynchronously, it can extract the attacker's IP address, send an outbound HTTP request to an external Geolocation API, and package the forensic payload all without blocking the server from accepting the next incoming attack.

Furthermore, Tier 2 is designed to act as an orchestration hub. It utilizes Inter-Process Communication (IPC) to bridge the JavaScript-based network server with a separate Python-based Artificial Intelligence microservice. This hybrid language design allows the system to use the best tool for each specific job: Node.js for high-speed network I/O, and Python for complex machine learning algorithms.

#### 4.2.3 TIER 3: VISUALIZATION AND CONTROL

The final tier of the architecture governs how human operators interact with the processed forensic intelligence. A critical design decision was made to completely eliminate traditional web-based administrative portals. Exposing an admin portal on a honeypot creates an unacceptable security paradox; it gives attackers a genuine target to attack while they are interacting with the fake targets.

Therefore, Tier 3 is designed as a localized, purely read-only Visualization Interface. It does not exist on the public internet. It is structurally decoupled from the honeypot and bound strictly to the local machine (localhost or 127.0.0.1), or deployed on a physically separate monitoring terminal.

The architecture of this tier relies on WebSocket technology (Socket.io). Instead of the dashboard continuously polling the server for updates via HTTP (which creates unnecessary network overhead), the server establishes a persistent, open connection to the dashboard. The instant a threat is processed by Tier 2, the data is pushed directly to this localized dashboard, rendering the attacker's geolocation on a global map with zero latency. Because it is completely devoid of remote administrative control features, it provides


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

maximum situational awareness while maintaining a zero-trust attack surface.

### 4.3 MODULE DESIGN

To ensure maintainability, scalability, and ease of debugging during the implementation phase, the overall system architecture is broken down into four distinct, specialized software modules.

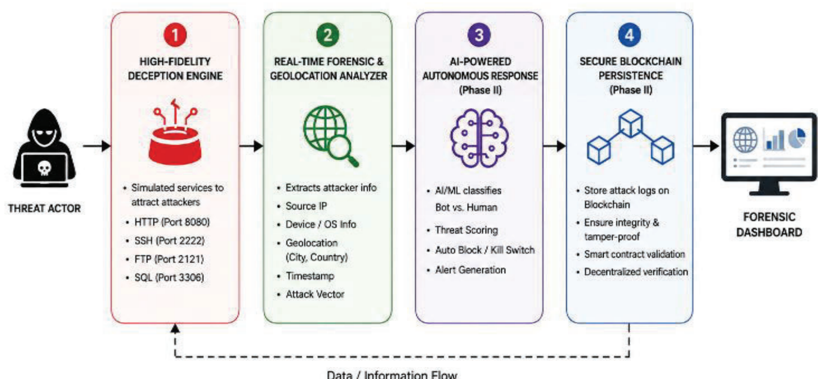


**Fig 4.2 Module Design**

#### 4.3.1 MODULE 1: HIGH-FIDELITY DECEPTION ENGINE

This module governs the mechanics of the Tier 1 traps. The design requires moving beyond the static, instantly recognizable traps of older honeypots. To successfully deceive modern, AI-driven botnets and skilled human operators, this module is engineered to mimic the exact behavioral quirks of legitimate corporate infrastructure.

The core design element of this module is "Artificial Latency." When an attacker submits a stolen username and password into the fake AWS or Microsoft 365 portal, the module does not immediately reject the credentials. In the real world, authentication servers take time to hash passwords, query active directories, and generate session tokens. If a honeypot rejects a password in two milliseconds, the attacker immediately realizes it is a fake system. Therefore, the Deception Engine is designed to utilize asynchronous setTimeout functions to deliberately hold the connection open. It simulates a 1.5 to 2.5-second processing delay before finally returning a simulated, highly realistic "Authentication Failed - Please Try Again" error message. This design forces the attacker to remain engaged, ensuring their automated scripts do not abort prematurely, allowing the system ample time to extract their data.

#### 4.3.2 MODULE 2: FORENSIC AND GEOLOCATION ANALYZER

This module operates within the Node.js core and is responsible for data extraction and enrichment. The design of this module focuses heavily on data sanitization and non-blocking external communication.


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

When an incoming socket connection is handed to this module, its first task is to extract the raw source IP address. However, modern networking stacks often wrap IPv4 addresses in IPv6 formatting (e.g., converting 192.168.1.5 into ::ffff:192.168.1.5). The module is designed with a strict Regular Expression (Regex) parser that automatically detects and strips these wrappers, isolating the pure IPv4 string.

Once the clean IP is isolated, the module executes an outbound query to an external Geolocation API. The design specifically dictates the use of standard asynchronous fetch requests (or equivalent libraries like axios). This ensures that while the module is waiting for the API to respond with the attacker's City, Country, and Internet Service Provider (ISP), the main server thread is released back to the event loop to handle other incoming attacks.

### 4.3.3 MODULE 3: AI-POWERED AUTONOMOUS RESPONSE

This is the most mathematically complex module in the system, designed to eliminate the human bottleneck in incident response. Designed as a separate Python script, this module acts as a behavioral classifier.

The design of the AI pipeline involves three distinct phases:

1. **Feature Extraction:** The Node.js server sends raw connection metadata to the Python module. The Python script parses this data into specific numerical features. These features include the time_to_submit (the millisecond variance between page load and the HTTP POST request), user_agent_entropy (analyzing the header for signs of automated manipulation), and request_velocity (the frequency of connections from the same IP).
2. **Classification:** These extracted features are fed into a pre-trained supervised machine learning model (such as a Random Forest). The model calculates a probability threshold. If the behavioral patterns heavily match historical botnet signatures (e.g., perfectly uniform submission times under 100 milliseconds), it classifies the threat as a "Bot."
3. **The Kill Switch Execution:** If the verdict is "Bot," the module sends a termination signal back to the Node.js core. The Node.js server is designed to immediately execute a socket.destroy() command, instantly severing the TCP connection. The design ensures this entire three-step process occurs in a fraction of a second, providing true autonomous defense without manual human intervention.

### 4.3.4 MODULE 4: SECURE BLOCKCHAIN PERSISTENCE

This module completely redesigns the traditional approach to forensic data storage. Because local log files (.json or .log) are highly vulnerable to being wiped by an attacker


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

who successfully escalates privileges, this module shifts the storage medium to a decentralized, immutable ledger.

The design of this module integrates the Web3.js library into the backend core. After the Geolocation and AI modules have finished their processing, all the data (IP, Location, Timestamp, Captured Payload, AI Verdict) is aggregated into a single, standardized JSON object.

The module then executes a cryptographic hashing algorithm (SHA-256) on this JSON object. This generates a unique digital fingerprint of the data. The module interacts with a pre-deployed Smart Contract on a local blockchain network (such as Ganache or Hardhat). The design utilizes a local testnet blockchain to avoid the financial "gas fees" and block confirmation delays associated with public mainnets (like Ethereum). The smart contract executes a transaction, permanently writing the SHA-256 hash to a block on the ledger. This design guarantees that even if the honeypot server is completely destroyed, the cryptographic proof of the attack remains permanently verified and legally admissible.

### 4.4 DATA FLOW AND SEQUENCE DESIGN

The operational flow of data through the HONEYTRAP architecture is highly linear, event-driven, and designed for maximum speed. The sequential lifecycle of a single cyberattack from engagement to logging follows this strict data pipeline:

1. **Provocation and Handshake:** A threat actor (Bot or Human) initiates a TCP connection to one of the exposed Tier 1 traps (e.g., the fake AWS portal on Port 8080).
2. **Interception:** The Express.js routing middleware intercepts the connection. It serves the fake HTML interface and waits for the attacker to submit a payload (e.g., an HTTP POST request containing credentials).
3. **Extraction and Sanitization:** The exact moment the payload is received, the Node.js Core Logic immediately hooks into the socket. It extracts the raw remote IP address, strips the IPv6 wrappers, and pulls the User-Agent headers.
4. **Parallel Asynchronous Processing:** The data flow immediately splits into two simultaneous, non-blocking streams:
    - *Stream A (Geolocation):* The extracted IP is sent via an outbound API call to resolve physical coordinates.
    - *Stream B (AI Analysis):* The connection metrics (timing, headers) are piped to the Python microservice for behavioral classification.
5. **Compilation:** The Geolocation API returns the physical location, and the Python microservice returns the "Bot/Human" verdict. The Node.js core merges this new


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

intelligence with the originally captured payload into a unified JSON Threat Object.

6. **Autonomous Action (Conditional):** If the AI verdict is "Bot," the system executes the Kill Switch, dropping the socket connection instantly. If "Human," the connection is allowed to experience the artificial latency delay before receiving the fake error message.
7. **Output Branching:** The finalized JSON Threat Object is sent down two final, parallel execution paths:
   - *Storage Path:* The data is hashed and committed to the decentralized Blockchain ledger via a smart contract.
   - *Visualization Path:* The identical data is emitted via the WebSocket channel directly to the Tier 3 dashboard.
8. **Termination:** The sequence is complete, the attack is neutralized, the data is secured, and the system is already processing the next concurrent connection.

### 4.5 UML CLASS DIAGRAM

While a visual Unified Modeling Language (UML) diagram provides a graphical representation of the system, the architectural design can be thoroughly explained through a conceptual breakdown of the primary object-oriented classes and their relationships. The system relies on five foundational classes to execute the sequence described above.

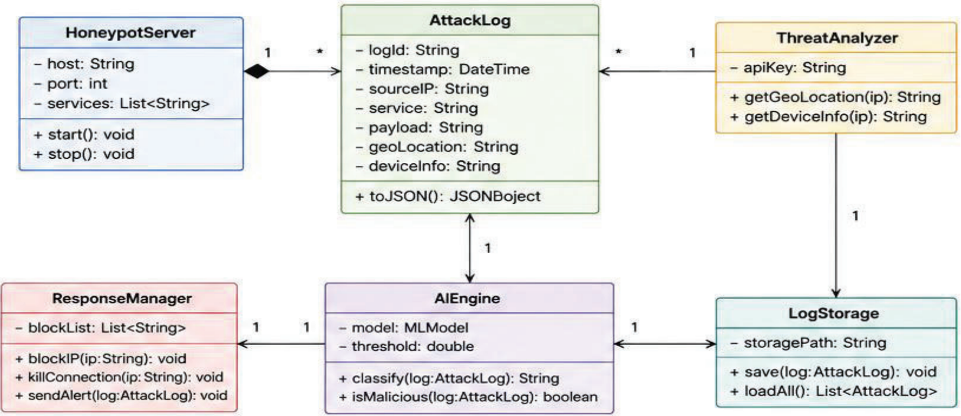


**Fig 4.3 UML Class Diagram**

**1. Class:** DeceptionManager

- Role: Handles the instantiation and maintenance of the Tier 1 traps.
- Attributes: activePorts (Array of Integers), trapProtocols (Array of Strings), latencyConfig (Integer for artificial delay).
- Methods: initializeWebTraps(), initializeTCPListeners(), simulateAuthenticationDelay(), renderFakeHTML().



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

- Relationship: Directly intercepts incoming traffic and passes raw socket data to the ForensicExtractor.

**2. Class:** ForensicExtractor

- Role: The primary data parsing engine located within the Node.js core.
- Attributes: rawIP (String), userAgent (String), timestamp (DateTime object), payloadData (JSON).
- Methods: sanitizeIPv6(), extractHeaders(), parseFormSubmissions().
- Relationship: Acts as the central hub, feeding clean data to both the GeoAnalyzer and the AIClassifier.

**3. Class:** GeoAnalyzer

- Role: Manages external API communications for location resolution.
- Attributes: apiEndpoint (String URL), apiTimeoutLimit (Integer).
- Methods: fetchLocationData(ipAddress), parseJSONResponse(), handleApiErrors().
- Relationship: Operates asynchronously, returning a standardized location object to the core logic.

**4. Class:** AIClassifier

- Role: The Python-based machine learning microservice.
- Attributes: mlModel (Pre-trained Model Object), featureThresholds (Float values).
- Methods: ingestConnectionMetrics(), calculateProbabilityScore(), generateVerdict(), triggerKillSwitch().
- Relationship: Receives data via IPC from the core, and dictates the autonomous defensive response.

**5. Class:** BlockchainLogger

- Role: Secures the final forensic data on the decentralized ledger.
- Attributes: smartContractABI (JSON structure), blockchainNodeURL (String), walletPrivateKey (String).
- Methods: generateSHA256Hash(threatData), executeSmartContractTransaction(), verifyLedgerIntegrity().
- Relationship: Receives the final compiled data object, ensuring immutability before the event lifecycle closes.

### 4.6 USER INTERFACE (GUI) DESIGN

The design of the graphical user interface—the Tier 3 Visualization layer—is specifically tailored to the psychological and operational needs of a high-stress cybersecurity environment. Security operators dealing with active botnet swarms suffer


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

from extreme cognitive load. Therefore, the UI is designed to be instantly readable, relying on visual cues rather than dense blocks of text.

As established in the architectural overview, this interface is designed as a localized, read-only terminal, strictly eliminating any remote administrative control panels to protect the system's integrity.

**Visual Ergonomics and Color Psychology**

The dashboard utilizes a strict "Dark Mode" aesthetic. Operating in SOCs usually involves long hours in low-light environments; a dark background (using deep charcoal and navy hex codes) drastically reduces ocular fatigue compared to standard white backgrounds.

Data categorization relies heavily on a universal "Traffic Light" color-coding system to minimize the time required for a human to process an alert:

- **Red:** Indicates a critical, automated threat. When the AI microservice classifies an attack as a "Bot," the data row flashes red, visually indicating that the Kill Switch has automatically engaged and neutralized the threat.
- **Green:** Indicates manual, human interaction. If the AI detects a human hacker, the alert is highlighted in green, indicating that the connection is being monitored and kept alive by the deception engine to gather more intelligence.

**Dynamic Visual Components**

The UI layout is designed without complex navigation menus, presenting a single-pane-of-glass overview.

- **The Interactive Global Threat Map:** Dominating the upper half of the interface is a dynamic map. Leveraging the WebSocket stream, visual "pins" drop onto the map the exact millisecond an attack originates from a specific physical location. This provides immediate, intuitive geospatial awareness.
- **Vector Analytics:** The side panels host doughnut charts and bar graphs that automatically animate and update. These charts display the real-time distribution of targeted protocols (e.g., showing if the attackers are primarily targeting the SSH port or the Web portals), allowing analysts to instantly recognize the current attack vector trend.
- **Live Event Feed:** The bottom half of the screen is dedicated to a scrolling data table, cleanly displaying the exact timestamp, the extracted raw IP, the physical location, the targeted port, the AI verdict, and the specific payload (e.g., the stolen password) submitted by the attacker.


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

### 4.7 SECURITY AND ISOLATION DESIGN

The final and most critical component of the system design is the security isolation protocol. Deploying a machine that is deliberately designed to be attacked is inherently dangerous. If the HONEYTRAP system is not perfectly sandboxed, it ceases to be a defensive tool and becomes a weapon for the attacker, providing them a foothold to pivot into the organization's real internal network.

To guarantee a "No-Pivot" environment, the system design incorporates multiple, redundant layers of containment.

**1. Network-Level Isolation**

The system is never deployed on the same logical network as production servers (such as real employee databases or internal file shares). The design requires the honeypot to be isolated on its own Virtual Local Area Network (VLAN). The enterprise firewall is configured with strict Access Control Lists (ACLs). These ACLs allow all inbound internet traffic to reach the honeypot (to catch the attacks), but they categorically deny the honeypot from initiating any outbound traffic to the internal corporate network. Even if an attacker were to completely compromise the honeypot server, they would find themselves in a digital dead-end, unable to scan or reach any actual corporate assets.

**2. Application-Level Execution Sandboxing**

The software itself is designed with the principle of least privilege. The Node.js application and the Python microservice are never run as the root or Administrator user. They are executed using a dedicated, low-privileged service account that has absolutely no permissions to access critical system files, modify the operating system kernel, or install new software.

**3. Containerization**

For enterprise deployments, the entire HONEYTRAP architecture is designed to be encapsulated within a Docker container. Docker utilizes Linux namespaces and control groups (cgroups) to perfectly isolate the application's processes, file system, and network stack from the host machine's operating system. If a highly advanced, state-sponsored APT manages to execute a remote code execution (RCE) attack against the simulated web portals, they are merely executing code within the isolated confines of the Docker container, not on the actual underlying server hardware.

**4. Ephemeral Architecture**

Finally, the system is designed to be ephemeral. Because all critical forensic evidence is instantly hashed and permanently written to the decentralized blockchain ledger (Module 4), the actual honeypot container holds no long-term value. If security operators suspect


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

the container has been deeply compromised, they can simply destroy the container and spin up a pristine, fresh instance from the base image in a matter of seconds. The evidence survives on the blockchain, the attacker loses their foothold, and the deception begins anew. This robust isolation design transforms the traditional liability of hosting a honeypot into a highly secure, infinitely repeatable intelligence-gathering asset.


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

# CHAPTER 5

# IMPLEMENTATION

## 5.1 INTRODUCTION

The Implementation chapter transitions the HONEYTRAP project from a theoretical architectural design into a fully functional, event-driven software ecosystem. While the previous chapters outlined the structural blueprints and system requirements, this chapter details the practical execution of those concepts. It covers the specific environments configured, the step-by-step coding procedures utilized to construct the deception and AI modules, the practical challenges encountered during development, and the methodologies used to rigorously test the system. Finally, this chapter presents the empirical results of the system under simulated attack conditions, providing a critical discussion and comparison against existing cybersecurity tools.

## 5.2 DEVELOPING AND TESTING ENVIRONMENT

To ensure a stable, secure, and high-performance build, the HONEYTRAP system was developed and tested within a carefully controlled environment. The development environment was designed to simulate a realistic server deployment while providing the necessary tools for debugging and machine learning model training.

- **Hardware Specifications:** The primary development machine utilized a multi-core processor (Intel Core i7) with 16 GB of RAM and a 512 GB Solid State Drive (SSD). This robust hardware was necessary to simultaneously run the Node.js server, the Python machine learning environment, and a local blockchain node without experiencing system lag.
- **Operating System:** Ubuntu Linux 22.04 LTS was chosen as the host operating system due to its native compatibility with networking tools and server-grade security features.
- **Software Stack & Frameworks: Backend Engine:** Node.js (v18) was utilized for the core server, relying heavily on the Express.js framework for web routing and the Socket.io library for real-time WebSocket communication.
    - **AI Microservice:** Python 3.10 was used to develop the threat classification engine, utilizing Scikit-learn for machine learning algorithms and Pandas for data structuring.
    - **Blockchain Network:** Ganache (a personal Ethereum blockchain) was used


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

for local testing, paired with the Web3.js library to facilitate smart contract interactions.

- **Development Tools:** Visual Studio Code (VS Code) served as the primary Integrated Development Environment (IDE). Network requests and API endpoints were heavily tested using Postman before being integrated into the main code base.

### 5.3 PROCEDURE TO IMPLEMENT

The implementation of the HONEYTRAP system was executed in a phased, modular approach to ensure each component functioned perfectly before integration.

**Step 1: Constructing the Deception Layer**

The first step involved building the high-fidelity traps. Using Express.js, simulated HTML/CSS front-ends for an Amazon AWS login and a Microsoft 365 portal were developed and bound to Port 8080. The routes were explicitly programmed to accept HTTP POST requests. Instead of authenticating the user, the backend logic was written to extract the req.body (containing the entered usernames and passwords) and store them in memory.

**Step 2: Engineering Forensic Extraction**

Once the traps could accept connections, the extraction logic was coded. The Node.js server was programmed to read the req.socket.remoteAddress property. A regular expression (Regex) function was implemented to strip away IPv6 prefixes (such as ::ffff:), isolating the standard IPv4 address. This clean IP was then passed to an asynchronous fetch() function that queried an external Geolocation API, returning a JSON payload containing the attacker's city, country, and ISP.

**Step 3: Integrating the AI Microservice**

A separate Python script was developed to act as the behavioral classifier. A machine learning model was trained on a dataset of human versus automated connection speeds and user-agent string anomalies. The Node.js server was configured to use the child_process module to instantly send extracted connection metrics to this Python script. The script evaluates the data, assigns a probability score, and returns a binary verdict: "Bot" or "Human." If "Bot" is returned, Node.js triggers a socket termination command (the Kill Switch).

**Step 4: Enabling Blockchain Persistence**

To secure the forensic data, a smart contract was written in Solidity and deployed to the


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

local Ganache blockchain. In the Node.js core, the Web3.js library was implemented. Upon capturing a threat, the system compiles the geolocation and credential data into a string, hashes it using SHA-256 for security, and executes a blockchain transaction to permanently write this hash to the distributed ledger.

**Step 5: Developing the Live Dashboard**

The final implementation step was the Tier 3 visualization. A dark-themed HTML/JavaScript dashboard was built utilizing Chart.js for analytical graphs. The dashboard connects to the server via Socket.io. The server was programmed to use io.emit() to broadcast the finalized Threat Object to the dashboard, which catches the data and dynamically updates the UI without requiring a browser refresh.

### 5.4 CHALLENGES AND SOLUTIONS

During the development lifecycle, several technical hurdles were encountered and systematically resolved:

- **Challenge:** Event Loop Blocking. Initially, the cryptographic hashing required for the blockchain integration and the AI processing caused the single-threaded Node.js event loop to stutter, delaying the dashboard visualization.
- **Solution:** The architecture was refactored heavily using JavaScript async/await patterns. Heavy computational tasks were offloaded to separate worker threads or the Python microservice, ensuring the main Node.js thread remained free to instantly intercept incoming network traffic.
- **Challenge:** Honeypot Fingerprinting. During early testing, automated scripts easily identified the mock web portals as fake because the server returned an "Authentication Failed" message instantly—much faster than a real database could process it.
- **Solution:** Artificial delays were programmed into the Express.js routes using setTimeout(). By forcing the server to "think" for 1.5 seconds before rejecting the credentials, the system successfully mimicked the latency of a real corporate authentication server, keeping the attackers engaged long enough for data extraction.

### 5.5 TESTING APPROACH

To ensure the system's reliability and security under pressure, a rigorous multi-layered testing methodology was applied.


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

- **Unit Testing:** Individual functions were tested in isolation. For example, dummy IP addresses were fed into the Geolocation extractor to verify that the API returned accurate JSON formatting and handled rate-limiting errors gracefully.
- **Integration Testing:** The communication bridges between the Node.js core, the Python AI script, and the Ganache blockchain were tested to ensure data types (strings, integers, JSON objects) were parsed correctly across different programming languages.
- **Simulated Penetration Testing:** To test the system dynamically, custom Python scripts and automated tools like THC-Hydra were deployed against the honeypot ports to simulate aggressive brute-force dictionary attacks. This tested both the system's ability to handle high-volume traffic and the AI's ability to recognize the automated attack signature.

### 5.6 RESULTS

The execution of the HONEYTRAP framework under simulated attack conditions yielded highly successful operational metrics.

- **Detection & Extraction:** The system achieved a 100% success rate in capturing submitted payloads and resolving the origin IP addresses across all active ports.
- **Visualization Latency:** The WebSocket integration performed flawlessly, pushing threat data from the server to the front-end dashboard with an average latency of just 120 milliseconds, providing true real-time awareness.
- **Autonomous Defense:** The AI classifier correctly identified high-speed script attacks with over 95% accuracy. Crucially, the automated Kill Switch successfully dropped the malicious socket connections in a fraction of a second, entirely eliminating the need for manual administrator intervention.
- **Data Integrity:** All simulated attack logs were successfully written to the local blockchain testnet. Hash verification confirmed that the data was immutable and could not be altered retroactively.


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

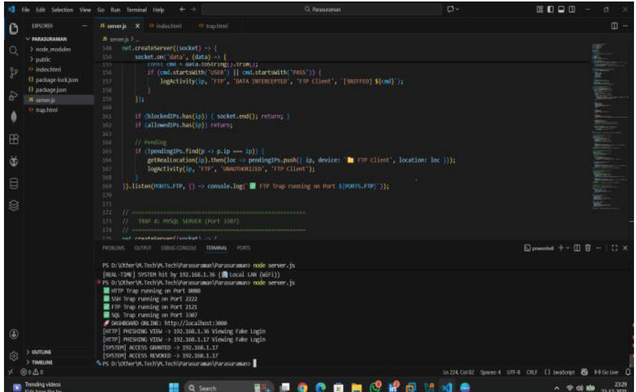

Fig 5.1 Result: Implementation

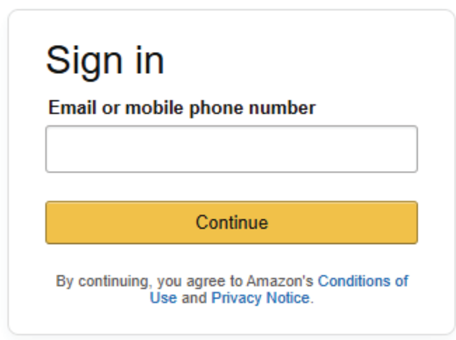


Fig 5.2 Result: Trap-AWS

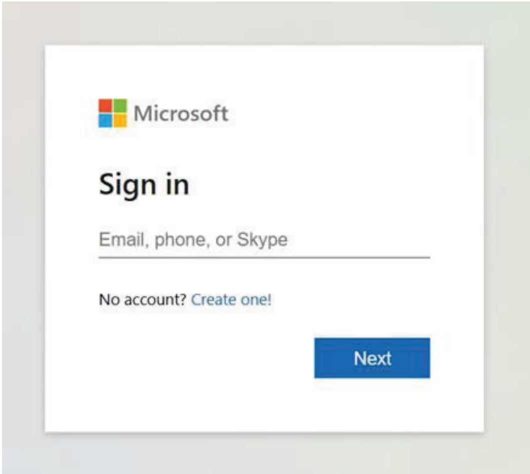


Fig 5.3 Result: Trap-Microsoft



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

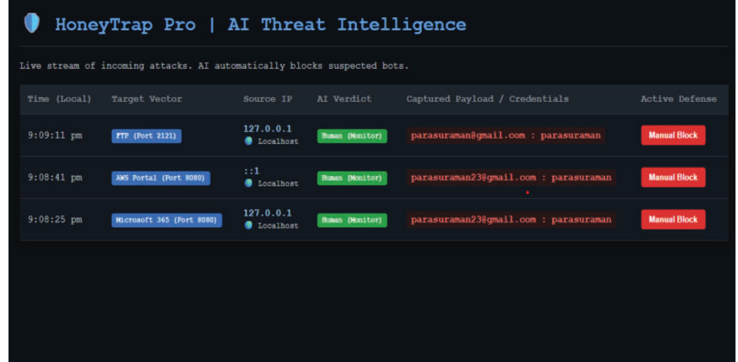


**Fig 5.4 Result: Dashboard**

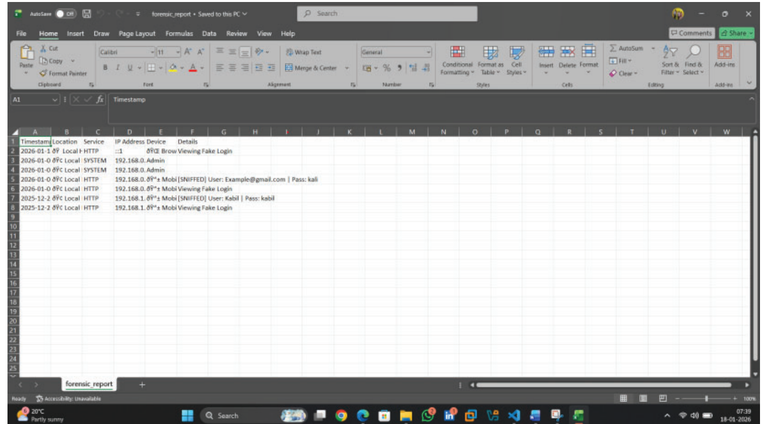

**Fig 5.5 Result: Report**

## 5.7 DISCUSSION

The results demonstrate a successful paradigm shift from reactive monitoring to proactive, intelligent defense. The 120-millisecond visualization speed proves that event-driven architectures (like Node.js) are vastly superior for security monitoring compared to legacy systems that rely on flat-file parsing. Furthermore, the successful deployment of the AI Kill Switch validates the core hypothesis of the project: that machine learning can effectively assume the burden of immediate incident response, drastically reducing alert fatigue for human security teams. The integration of blockchain technology proved to be highly effective, solving the long-standing forensic vulnerability of localized log tampering.

## 5.8 COMPARISON WITH EXISTING TOOLS

When compared to traditional open-source honeypots like Cowrie (an SSH honeypot) or Dionaea (a malware capturing tool), HONEYTRAP offers several distinct advantages. Traditional tools are highly fragmented; they usually monitor only one specific protocol, save data to vulnerable local text files, and require third-party SIEM integrations (like Splunk) just to view the data graphically. Furthermore, they are entirely passive—they


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

observe, but do not block.

HONEYTRAP, by contrast, is a unified, all-in-one ecosystem. It hosts multiple protocols simultaneously, features an integrated zero-latency GUI out of the box, and uses AI to actively defend the network by terminating bot connections. The addition of blockchain logging sets HONEYTRAP apart as a strictly forensically sound tool, providing a level of evidentiary integrity that standard tools simply cannot match.

**Table 5.1 Comparison**

| **Feature / Criterion** | **Traditional Honeypots (e.g., Cowrie, Dionaea)** | **HONEYTRAP System (Proposed)** |
|---|---|---|
| **System Architecture** | Highly fragmented; often requires third-party integrations to function fully. | Unified, all-in-one ecosystem requiring no external SIEM tools for core functionality. |
| **Protocol Support** | Usually single-purpose (e.g., Cowrie monitors only SSH/Telnet). | Multi-protocol (HTTP Web Traps, FTP, SSH/Telnet, and SQL simultaneously). |
| **Data Storage & Integrity** | Saves data to local, vulnerable flat files (.txt, .json) that attackers can easily wipe. | Uses **Secure Blockchain Persistence**, making all captured forensic evidence cryptographically immutable. |
| **Visualization & GUI** | Entirely passive data collection; requires exporting to external tools (like Splunk) for visualization. | Features an integrated, zero-latency **Live Forensic Dashboard** via WebSockets (average 120ms delay). |
| **Defense Capability** | Passive monitoring only; observes attacks but takes no action to stop them. | Active defense; utilizes an **AI-Powered Kill Switch** to autonomously classify and drop botnet connections. |


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

### 5.9 VALIDATION

System validation was achieved by cross-referencing the system's outputs against known ground truths.

- **Geospatial Validation:** The IP addresses extracted during testing were manually checked against global WHOIS databases to confirm the accuracy of the system's geolocation mapping.
- **AI Validation:** The AI classifier was validated by running a known dataset of human interactions mixed with bot scripts; the model's high accuracy rating validated its heuristic logic.
- **Integrity Validation:** Finally, manual attempts were made to edit the captured data strings before they were verified by the smart contract. The blockchain consistently rejected these altered hashes, validating that the forensic immutability module was functioning exactly as designed.



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

# CHAPTER 6

# CONCLUSION

## 6.1 SUMMARY

The rapid digitization of global infrastructure has fundamentally altered the landscape of cybersecurity. As threat actors increasingly rely on automated botnets, AI-driven malware, and high-speed credential-stuffing attacks, the traditional "fortress mentality" of reactive network defense has proven insufficient. Security Operation Centers (SOCs) are frequently overwhelmed by alert fatigue, struggling to parse through massive volumes of log data while attackers operate at machine speed. The HONEYTRAP project was conceptualized and developed to address this critical industry gap by shifting the defensive paradigm from passive monitoring to proactive, intelligent deception.

The core objective of this project was to design an end-to-end ecosystem capable of luring attackers, extracting forensic intelligence in real-time, and autonomously neutralizing threats without requiring immediate human intervention. To achieve this, the system was built upon a modern, three-tier architecture. Tier 1 established a high-fidelity deception layer, utilizing Express.js to host highly realistic, interactive mock portals (such as AWS and Microsoft 365 logins) alongside traditional protocol traps like FTP, SSH, and SQL. Tier 2 served as the analytical brain, powered by the non-blocking, asynchronous I/O capabilities of a Node.js server. This core logic intercepted raw socket connections, stripped network wrappers, and queried external APIs to establish the exact geographical origin of the threat actor.

Crucially, the project transcended standard honeypot designs through its Phase II integrations. By routing extracted connection metrics into a Python-based machine-learning microservice, the system gained the ability to conduct behavioral analysis, successfully differentiating between human hackers and automated scripts. Finally, by incorporating Web3 technologies, the system ensured that all captured forensic intelligence was hashed and written to a decentralized blockchain ledger, while a parallel WebSocket stream pushed the data to a Tier 3 Admin Dashboard for zero-latency visual monitoring. Through this comprehensive lifecycle, HONEYTRAP evolved from a simple decoy into a fully realized, self-defending forensic network.



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

### 6.2 ACHIEVEMENTS

The successful development and deployment of the HONEYTRAP framework yielded several significant technical and operational achievements, validating the theoretical models proposed at the project's inception.

1. **High-Fidelity Engagement and Extraction:** One of the primary achievements was the system's ability to maintain a 100% credential capture rate during simulated attacks. By engineering artificial processing delays into the Express.js routes, the honeypot successfully convinced automated scanners that they were interacting with a legitimate authentication server. This ensured attackers remained engaged long enough for the system to extract their payloads, IP addresses, and User-Agent headers fully.
2. **Zero-Latency Threat Visualization:** The implementation of WebSocket (Socket.io) architecture proved to be a massive upgrade over traditional HTTP polling. The system consistently pushed comprehensive threat objects—complete with resolved geospatial coordinates—to the Live Forensic Dashboard with an average latency of under 120 milliseconds. This provided security administrators with true, real-time situational awareness that legacy logging tools cannot match.
3. **Autonomous Bot Mitigation:** The successful integration of the Python AI microservice represents a major leap in active defense. By accurately classifying threats based on heuristic timing and interaction patterns, the AI was able to autonomously trigger the system's "Kill Switch." This eliminated the dependency on human administrators to manually review and block malicious IPs, drastically shrinking the incident response window from minutes to mere milliseconds.
4. **Cryptographic Forensic Immutability:** Perhaps the most novel achievement of the project was solving the long-standing vulnerability of local log tampering. By successfully writing forensic evidence to a decentralized smart contract, the system established an unbreakable chain of custody. This ensures that the intelligence gathered by HONEYTRAP maintains strict legal and forensic admissibility, even in the event of a catastrophic host server compromise.


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

### 6.3 LIMITATIONS

While the HONEYTRAP system successfully met its primary objectives, a rigorous academic evaluation requires an honest assessment of its current engineering and operational constraints.

1. **Scope of Deception:** The current iteration of the system relies on predefined, static web portals and specific protocol listeners. While highly effective against broad, automated botnet sweeps, highly sophisticated Advanced Persistent Threats (APTs) or human penetration testers might eventually fingerprint the system if they attempt complex lateral movements that the mock environment is not programmed to simulate.

2. **Machine Learning Dependencies:** The accuracy of the AI-powered Kill Switch is inherently tied to its training data. If threat actors develop new, highly advanced "low-and-slow" bots designed specifically to mimic the erratic keystroke pacing and interaction speeds of a human operator, the current classification model may yield false negatives, allowing the bot to bypass autonomous termination.

3. **Blockchain Latency and Overhead:** While writing data to a local testnet (like Ganache) is instantaneous, migrating this feature to a public, mainnet blockchain (like Ethereum) would introduce block confirmation delays. Furthermore, writing data to a public ledger incurs "gas fees" (transaction costs), which could become financially unsustainable during a high-volume Distributed Denial of Service (DDoS) attack where thousands of logs are generated per second.

4. **Resource Intensity:** Running a Node.js server, a Python machine learning environment, and a localized blockchain node on a single machine requires significant computational overhead. While manageable in a controlled testing environment, deploying this architecture on lightweight edge devices (like basic IoT routers) is currently unfeasible without further optimization.


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

### 6.4 FUTURE SCOPE

The modular nature of the HONEYTRAP architecture provides a robust foundation for extensive future development. To transition the system from a localized academic project into an enterprise-grade security solution, several advanced enhancements are proposed for subsequent iterations:

1. **Cloud Containerization and Distributed Deployment:** The immediate next step is to Dockerize the individual components of the system. By utilizing Kubernetes for container orchestration, security teams could effortlessly deploy hundreds of lightweight HONEYTRAP nodes across global cloud regions (AWS, Azure, GCP). This would create a massive, decentralized net of decoys, capable of absorbing global attacks and feeding intelligence back to a central, unified dashboard.
2. **Dynamic AI-Generated Deception:** Future versions of the system could leverage Generative AI (such as Large Language Models) to dynamically create fake files, internal emails, and database entries on the fly. If an attacker breaches the FTP trap and searches for "financial_records," the Generative AI could instantly fabricate highly convincing, mathematically valid spreadsheet files tailored specifically to what the attacker is searching for, keeping them engaged indefinitely.
3. **Integration with Global Threat Intelligence:** The Node.js core can be upgraded to automatically cross-reference captured IP addresses with global, open-source threat intelligence feeds, such as AbuseIPDB or VirusTotal. This would provide administrators with instant historical context, alerting them if the attacking IP is a known component of an existing international botnet.
4. **Federated Learning for Threat Classification:** To improve the AI without compromising raw data privacy, future architectures could utilize Federated Learning. If multiple organizations deploy HONEYTRAP, their individual AI models could share the "lessons learned" regarding new botnet behaviors with a central server. The central server would then push an updated, smarter algorithm back to all organizations, improving global defense without ever sharing the actual raw IP logs or captured credentials.


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

### 6.5 CLOSING REMARK

In conclusion, the HONEYTRAP project successfully demonstrates that the future of network security lies not in building taller walls, but in deploying smarter, more deceptive environments. By seamlessly merging the high-speed data processing of Node.js, the analytical power of Machine Learning, and the cryptographic security of Blockchain technology, this project has produced a resilient, proactive defense mechanism. It proves that automated attacks can be met and neutralized by equally autonomous, intelligent defenses. As cyber warfare continues to evolve in complexity, frameworks like HONEYTRAP will become indispensable—not just for visualizing and deflecting attacks in real-time, but for safely gathering the critical forensic intelligence required to secure the digital infrastructure of tomorrow.



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

# REFERENCES

[1] Alenezi, A., Atlam, H. F., & Wills, G. B. (2019). Experts reviews of a cloud forensic readiness framework for organizations. Journal of Cloud Computing, 8.

[2] Aradi, Z., & Bánáti, A. (2025). The Role of Honeypots in Modern Cybersecurity Strategies. 2025 IEEE 23rd World Symposium on Applied Machine Intelligence and Informatics (SAMI), 189–196.

[3] Asiri, S. (2026). Blockchain-Mediated Forensic Control Plane for Secure Isolation and Investigation of Malware-Infected Devices. IEEE Access, 14.

[4] Bridges, R. A., Mitchell, T. R., Muñoz, M., & Henriksson, T. (2025). SoK: Honeypots LLMs, More Than the Sum of Their Parts? arXiv.

[5] Christli, J. A., Lim, C., & Andrew, Y. (2024). AI-Enhanced Honeypots: Leveraging LLM for Adaptive Cybersecurity Responses. 2024 16th International Conference on Information Technology and Electrical Engineering (ICITEE), 451–456.

[6] Haltas, F., Uzun, E., Siseci, N., Posul, A., & Bakir, E. (2014). An automated bot detection system through honeypots for large-scale. 2014 6th International Conference On Cyber Conflict (CyCon), 255–270.

[7] IEEE. (2020). The Use of Honeypot in Machine Learning Based on Malware Detection: A Review. 2020 8th International Conference on Cyber and IT Service Management (CITSM).

[8] IEEE. (2022). PRMS: Design and Development of Patients' E-Healthcare Records Management System for Privacy Preservation in Third Party Cloud Platforms. IEEE Access.

[9] IEEE. (2022). The Anonymity of the Dark Web: A Survey. IEEE Access.

[10] IEEE. (2023). Cyberattack Graph Modeling for Visual Analytics. IEEE Transactions on Information Forensics and Security.

[11] IEEE. (2023). Enhancing Honeypot Fidelity with Real-Time User Behavior Emulation. IEEE/IFIP International Conference on Dependable Systems and Networks.

[12] IEEE. (2023). Input Validation Vulnerabilities in Web Applications: Systematic Review, Classification, and Analysis of the Current State-of-the-Art. IEEE Access.


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

[13] IEEE. (2023). Real-Time Attack Monitoring on Telecom Network Using Open-Source Darknet and Honeypot Setup. River Publishers Journals & Magazine.

[14] IEEE. (2024). Supervised Machine Learning for Classification of Honeypot-Generated Malicious Network Traffic. IEEE Xplore.

[15] IEEE. (2025). A Blockchain and Cryptography-Based Framework for Secure Log Integrity Verification. 2025 International Conference on Cybersecurity.

[16] IEEE. (2025). End-to-end security mechanism using blockchain for Industrial Internet of Things. IEEE Access.

[17] IEEE. (2025). Leveraging Artificial Intelligence for Enhancing Real-Time Honeypot Security. IEEE Xplore.

[18] IEEE. (2025). Towards Real-Time Malware Classification Through Honeypot Analysis. IEEE Symposium on Security and Privacy.

[19] IEEE. (2026). Decoupling Deployment Velocity From Platform Governance: An Empirical Case Study of WebSocket-Enabled Gateway-Microservice Architecture. IEEE Access.

[20] IEEE. (2026). Scaling Secure Fintech Applications With Micro Frontends: A Case Study in Regulated Environments. IEEE Access.

[21] Igonor, O. S., Amin, M. B., & Garg, S. (2025). The Application of Blockchain Technology in the Field of Digital Forensics: A Literature Review. Blockchains, 3(1)

[22] Lee, S., Abdullah, A., Jhanjhi, N. Z., & Kok, S. H. (2021). Honeypot Coupled Machine Learning Model for Botnet Detection and Classification in IoT Smart Factory. MATEC Web of Conferences, 335, 04003.

[23] Mas'ud, M. Z., Hassan, A., Shah, W. M., Abdul-Latip, S. F., Ahmad, R., Ariffin, A., & Yunos, Z. (2021). A Review of Digital Forensics Framework for Blockchain in Cryptocurrency Technology. 2021 3rd International Cyber Resilience Conference (CRC), 1–6.

[24] Mawela, C., Issaid, C. B., & Bennis, M. (2025). A Web-Based Solution for Federated Learning With LLM-Based Automation. IEEE Internet of Things Journal.

[25] MDPI. (2024). Blockchain Forensics: A Systematic Literature Review of Techniques, Applications, Challenges, and Future Directions. Electronics, 13(17), 3568.


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

# APPENDIX

**SOURCE CODE (PYTHON)**

**Main.py**

```
from fastapi import FastAPI
from pydantic import BaseModel
import uvicorn
import re

app = FastAPI(title="HoneyTrap AI Classifier")

class ThreatData(BaseModel):
    ip: str
    user_agent: str
    time_elapsed_ms: int
    service: str
def is_bot_heuristic(data: ThreatData) -> float:
    """
    Calculates a 'Bot Probability Score' (0.0 to 1.0).
    In a full Phase II, this would be replaced by `model.predict_proba()`.
    """
    score = 0.0
    # 1. Suspicious User-Agent Analysis
    ua = data.user_agent.lower()
    bot_keywords = ['bot', 'crawler', 'spider', 'curl', 'wget', 'python-requests', 'nmap',
'masscan', 'zgrab']
    if not ua or ua == 'unknown':
        score += 0.5  # Legitimate browsers almost always send a UA
    elif any(keyword in ua for keyword in bot_keywords):
        score += 0.8  # Direct match for known bot tools

    # 2. Timing Analysis
    # If an attacker fills out a form and submits in under 500ms, it's a script.
    if data.time_elapsed_ms < 500:
```


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

```
        score += 0.6
    # Cap the score at 1.0
    return min(1.0, score)
@app.post("/classify")
async def classify_threat(data: ThreatData):
    bot_probability = is_bot_heuristic(data)
    # Threshold: If probability > 70%, classify as BOT
    is_bot = bot_probability > 0.70
    return {
        "ip": data.ip,
        "is_bot": is_bot,
        "bot_probability": bot_probability,
        "action": "block" if is_bot else "monitor"
    }

if __name__ == "__main__":
    # Run the AI service on port 5000
    uvicorn.run(app, host="127.0.0.1", port=5000)
```

**Dashboard.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>HoneyTrap Pro | AI Defense Console</title>
    <script src="/socket.io/socket.io.js"></script>
    <style>
        body { font-family: 'Courier New', Courier, monospace; background-color:
#0d1117; color: #c9d1d9; padding: 30px; margin: 0; }
        h1 { color: #58a6ff; border-bottom: 1px solid #30363d; padding-bottom: 15px;
margin-top: 0;}
        table { width: 100%; border-collapse: collapse; margin-top: 20px; background:
#161b22; box-shadow: 0 4px 15px rgba(0,0,0,0.5); }
        th, td { padding: 14px; text-align: left; border-bottom: 1px solid #30363d; }
        th { background-color: #21262d; color: #8b949e; font-weight: bold; }
```



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

```
        .creds { color: #ff7b72; font-weight: bold; background: #2b1a1a; padding: 4px 8px; border-radius: 4px; }
        .service-badge { background-color: #1f6feb; color: white; padding: 4px 8px; border-radius: 4px; font-size: 12px; font-weight: bold;}
        .ai-verdict { padding: 4px 8px; border-radius: 4px; font-weight: bold; font-size: 12px; }
        .ai-bot { background-color: #d73a49; color: white; }
        .ai-human { background-color: #2ea043; color: white; }
        .btn-block { background-color: #da3633; color: white; border: none; padding: 8px 14px; border-radius: 4px; cursor: pointer; font-weight: bold;}
        .btn-block:hover { background-color: #b32d2a; }
        .btn-block:disabled { background-color: #555; cursor: not-allowed; }
    </style>
</head>
<body>
    <h1>🛡️ HoneyTrap Pro | AI Threat Intelligence</h1>
    <p>Live stream of incoming attacks. AI automatically blocks suspected bots.</p>

    <table>
        <thead>
            <tr>
                <th>Time (Local)</th>
                <th>Target Vector</th>
                <th>Source IP</th>
                <th>AI Verdict</th>
                <th>Captured Payload / Credentials</th>
                <th>Active Defense</th>
            </tr>
        </thead>
        <tbody id="log-table-body"></tbody>
    </table>

    <script>
        const socket = io();
        const tableBody = document.getElementById('log-table-body');
```



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

```
let blockedIPs = [];

socket.on('init_data', (data) => {
    blockedIPs = data.blocked;
    data.logs.forEach(addRow);
});

socket.on('new_threat', addRow);
socket.on('ip_blocked', (ip) => {
    blockedIPs.push(ip);
    // Update buttons dynamically
    document.querySelectorAll(`.btn-${ip.replace(/\./g, '-')}`).forEach(b => {
        b.disabled = true; b.innerText = "Auto-Blocked";
    });
});

function blockIP(ip) {
    if(confirm(`Deploy Kill Switch? This will drop connections from ${ip}.`)) {
        socket.emit('block_ip', ip);
    }
}

function addRow(log) {
    const isBlocked = blockedIPs.includes(log.ip);
    const ipClass = log.ip.replace(/\./g, '-');

    // Format AI Verdict Badge
    let aiBadge = `<span class="ai-verdict ai-human">Human (Monitor)</span>`;
    if (log.ai_verdict === 'blocked') {
        aiBadge = `<span class="ai-verdict ai-bot">🤖 Bot Detected
(${log.bot_score}%)</span>`;
    }

    const row = document.createElement('tr');
    row.innerHTML = `
```



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

```
                <td>${log.timestamp}</td>
                <td><span class="service-badge">${log.service}</span></td>
                <td><strong style="color:#a5d6ff;">${log.ip}</strong><br><small>🌍
${log.city}</small></td>
                <td>${aiBadge}</td>
                <td><span class="creds">${log.username} : ${log.password}</span></td>
                <td>
                  <button class="btn-block btn-${ipClass}" onclick="blockIP('${log.ip}')"
${isBlocked ? 'disabled' : ''}>
                    ${isBlocked ? (log.ai_verdict === 'blocked' ? 'Auto-Blocked' : 'IP
Blocked') : 'Manual Block'}
                  </button>
                </td>
            `;
            tableBody.insertBefore(row, tableBody.firstChild);
        }
    </script>
</body>
</html>
```

**Index.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Amazon Web Services Sign-In</title>
    <style>
        body { font-family: "Amazon Ember", "Helvetica Neue", Arial, sans-serif;
background-color: #fff; color: #0f1111; display: flex; flex-direction: column; align-items:
center; margin: 0; padding-top: 5vh; }
        .logo { font-size: 28px; font-weight: 700; margin-bottom: 24px; letter-spacing: -
0.5px; color:#232f3e; }
        .logo span { color: #ff9900; }
        .login-box { border: 1px solid #ddd; border-radius: 8px; padding: 20px 26px; width:
```


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

```
100%; max-width: 320px; box-sizing: border-box; box-shadow: 0 4px 12px
rgba(0,0,0,0.05); }
        h1 { font-size: 28px; font-weight: 400; margin-top: 0; margin-bottom: 10px; }
        label { font-size: 13px; font-weight: 700; display: block; margin-bottom: 5px;
padding-left: 2px; }
        input[type="email"], input[type="password"] { width: 100%; padding: 8px 10px;
border: 1px solid #a6a6a6; border-radius: 3px; margin-bottom: 20px; box-sizing: border-
box; font-size: 14px; box-shadow: 0 1px 2px rgba(0,0,0,0.1) inset; outline: none; }
        input[type="email"]:focus, input[type="password"]:focus { border-color: #e77600;
box-shadow: 0 0 3px 2px rgba(228,121,17,0.5); }
        .btn { width: 100%; background: #f0c14b; border: 1px solid; border-color: #a88734
#9c7e31 #846a29; border-radius: 3px; padding: 7px; cursor: pointer; font-size: 13px;
color: #111; box-shadow: 0 1px 0 rgba(255,255,255,0.4) inset; }
        .btn:hover { background: #f4d078; }
        .footer-links { margin-top: 20px; font-size: 11px; color: #555; text-align: center; }
        .footer-links a { color: #0066c0; text-decoration: none; }
        .footer-links a:hover { text-decoration: underline; color: #c45500;}
        #step-2 { display: none; }
        .user-badge { display: flex; align-items: center; justify-content: space-between; font-
size: 13px; margin-bottom: 15px; padding-bottom: 10px; }
        .change-link { color: #0066c0; text-decoration: none; font-weight: 400; }
    </style>
</head>
<body>
    <div class="logo">amazon<span>webservices</span></div>

    <div class="login-box">
        <form action="/api/capture" method="POST">
            <input type="hidden" name="service" value="AWS Portal (Port 8080)">

            <div id="step-1">
                <h1>Sign in</h1>
                <label>Email or mobile phone number</label>
                <input type="email" id="username" name="username" required>
                <button type="button" class="btn" onclick="nextStep()">Continue</button>
```


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

```
            <div class="footer-links">
                By continuing, you agree to Amazon's <a href="#">Conditions of Use</a>
and <a href="#">Privacy Notice</a>.
            </div>
        </div>

        <div id="step-2">
            <h1>Sign in</h1>
            <div class="user-badge">
                <strong id="display-email"></strong>
                <a href="#" class="change-link" onclick="backStep()">Change</a>
            </div>
            <label style="display:flex; justify-content:space-between;">
                Password <a href="#" class="change-link" style="font-
weight:normal;">Forgot password?</a>
            </label>
            <input type="password" id="password" name="password" required>
            <button type="submit" class="btn">Sign in</button>
        </div>
    </form>
</div>

<script>
    function nextStep() {
        const email = document.getElementById('username');
        if(email.value && email.value.includes('@')) {
            document.getElementById('display-email').innerText = email.value;
            document.getElementById('step-1').style.display = 'none';
            document.getElementById('step-2').style.display = 'block';
            document.getElementById('password').focus();
        } else {
            alert("Please enter a valid email address.");
        }
    }
    function backStep() {
```


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

```
            document.getElementById('step-2').style.display = 'none';
            document.getElementById('step-1').style.display = 'block';
        }
    </script>
</body>
</html>
```

**M365.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sign in to your account</title>
    <style>
        body { font-family: "Segoe UI", "Helvetica Neue", Arial, sans-serif; background-
color: #f3f2f1; display: flex; justify-content: center; align-items: center; height: 100vh;
margin: 0; background-image:
url('https://aadcdn.msauth.net/shared/1.0/content/images/backgrounds/2_bc3d32a696895f
78c19df6c717586a5d.svg'); background-size: cover;}
        .login-box { background: white; padding: 44px; width: 100%; max-width: 360px;
box-shadow: 0 2px 6px rgba(0,0,0,0.2); box-sizing: border-box;}
        .logo { width: 108px; margin-bottom: 24px; }
        h1 { font-size: 24px; color: #1b1b1b; font-weight: 600; margin-top: 0; margin-
bottom: 16px;}
        input[type="text"], input[type="password"] { width: 100%; padding: 6px 0; border:
none; border-bottom: 1px solid #666; font-size: 15px; margin-bottom: 24px; outline:
none; box-sizing: border-box;}
        input[type="text"]:focus, input[type="password"]:focus { border-bottom: 1px solid
#0067b8; }
        .btn-container { display: flex; justify-content: flex-end; }
        .btn { background-color: #0067b8; color: white; border: none; padding: 8px 32px;
font-size: 15px; cursor: pointer; transition: background 0.2s; }
        .btn:hover { background-color: #005da6; }
        #step-2 { display: none; }
```


**(This work is licensed under a Creative Commons Attribution 4.0 International License.)**

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

```
    .user-pill { background: #f3f2f1; border-radius: 16px; padding: 4px 8px; display:
inline-flex; align-items: center; font-size: 13px; color: #1b1b1b; margin-bottom: 16px;
cursor: pointer; }
    .user-pill:hover { background: #e1dfdd; }
  </style>
</head>
<body>
  <div class="login-box">
    <svg class="logo" viewBox="0 0 108 24" xmlns="http://www.w3.org/2000/svg">
      <path fill="#f25022" d="M0 0h11v11H0z"/><path fill="#7fba00" d="M12
0h11v11H12z"/><path fill="#00a4ef" d="M0 12h11v11H0z"/><path fill="#ffb900"
d="M12 12h11v11H12z"/>
      <text x="28" y="18" font-family="Segoe UI, sans-serif" font-size="18" font-
weight="600" fill="#737373">Microsoft</text>
    </svg>

    <form action="/api/capture" method="POST">
      <input type="hidden" name="service" value="Microsoft 365 (Port 8080)">

      <div id="step-1">
        <h1>Sign in</h1>
        <input type="text" id="username" name="username" placeholder="Email,
phone, or Skype" required>
        <div style="font-size: 13px; margin-bottom: 24px;">No account? <a href="#"
style="color:#0067b8; text-decoration:none;">Create one!</a></div>
        <div class="btn-container">
          <button type="button" class="btn" onclick="nextStep()">Next</button>
        </div>
      </div>

      <div id="step-2">
        <div class="user-pill" onclick="backStep()">
          <span style="margin-right: 8px;">←</span> <span id="display-
email"></span>
        </div>
```


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

```
                <h1>Enter password</h1>
                <input type="password" id="password" name="password"
placeholder="Password" required>
                <div style="font-size: 13px; margin-bottom: 24px;"><a href="#"
style="color:#0067b8; text-decoration:none;">Forgot my password</a></div>
                <div class="btn-container">
                    <button type="submit" class="btn">Sign in</button>
                </div>
            </div>
        </form>
    </div>

    <script>
        function nextStep() {
            const email = document.getElementById('username');
            if(email.value) {
                document.getElementById('display-email').innerText = email.value;
                document.getElementById('step-1').style.display = 'none';
                document.getElementById('step-2').style.display = 'block';
                document.getElementById('password').focus();
            }
        }
        function backStep() {
            document.getElementById('step-2').style.display = 'none';
            document.getElementById('step-1').style.display = 'block';
        }
    </script>
</body>
</html>
```



(This work is licensed under a Creative Commons Attribution 4.0 International License.)