Project Specification

AI Enabled Honeypot

Conor Hendley
C00257507

Bachelor of Science (Honours)
in Cybercrime and IT Security

Supervisor: Christopher Staff

Table of Contents

1.0 Introduction

1.1 Background

A honeypot is a security mechanism that is purposely designed to masquerade as a vulnerable system to malicious actors. They are utilised in attempts to lure attackers away from an organisations or individual's live system and keep them distracted, thus keeping the real system out of harm's way. Honeypots also allow security professionals to observe an attacker's behaviour in a controlled environment. Through the simulated system, defenders can monitor an attacker's behaviour, gaining valuable insight into their techniques, tools, and potential motivation, all with minimal risk to the live system.

1.2 Problem Statement

Despite the value that honeypots provide organisations and defence experts, many are limited by their predictability. An experienced attacker can quickly fingerprint a common honeypot framework. This is often achieved by analysing network behaviour, system configurations, and most obviously through observing inconsistent or incorrect responses from the system. This can result in short interaction times between the attacker and the honeypot, leading to a minimal amount of intelligence gathered.

This lack of adaptability to the attackers' actions on the system undermines an incredibly important purpose of honeypots: to gather in-depth behavioural data on malicious actors. Without this extended engagement data, security experts miss out on the opportunity to study an attacker's tactics, persistence strategies, escalation methods, and potential payload deployment. As such there is a real need for honeypots to not only become harder to detect through more sophisticated configurations but also to be able to convince the attacker they are in a real system by responding appropriately and dynamically to the attackers' actions.

1.3 Objectives

This project aims to design and develop an AI Enabled Honeypot that leverages artificial intelligence to generate realistic, context-aware system behaviours and responses to a malicious actors' actions. By doing so, the project aims to extend attacker interaction time with the honeypot, this should ideally correlate to a greater quantity and quality of behavioural data collected.

1.4 Project Scope

This Project will focus on the development of a honeypot capable of taking in commands from an attacker and then utilizing an LLM to generate a contextually aware and believable output. The honeypot will contain the following components:

- Command Handling Layer: This will be used to capture the attackers' input so it can be sent to the AI Response Engine.
- AI Response Engine: In the background to generate appropriate and believable response to the attacker's input.

- Emulated CLI: The honeypot will present the attacker with an emulated CLI that they can interact with and send commands through.
- Logging and Data Collection Module: To record and log all activity on the honeypot for later analysis.
- Baseline Model for Comparison: A simple static honeypot implementation to allow for comparison data to be acquired.

This project will be limited to creating a research prototype suitable for controlled testing. It will not include deployment to any live network or large scale honeynet. The focus will be on creating a proof of concept to demonstrate that a honeypot can be created with AI integration to potentially extend attacker engagement for better quality data collection.

2.0 System Overview

2.1 Concept

The AI Enabled Honeypot is designed to act as a decoy system that will engage malicious actors through realistic, context aware interactions. Unlike with static honeypots that rely on predefined scripted responses, the AI Enabled Honeypot will integrate an AI Response Engine that utilizes a LLM that has been trained to generate believable dynamic outputs to the attackers' commands. The goal of the AI Enabled Honeypot is to maintain a believable illusion of authenticity for extended periods of time, this will encourage deeper and longer engagement from the attacker. This in turn should result in an increase in both the quality and quantity of behavioural data that could be collected for future analysis. This project will aim to serve as a proof of concept. It will demonstrate how an AI could be integrated into a honeypot framework to provide realistic and believable responses. As previously mentioned, the honeypot will consist of several key components, these being: A command handling layer, a trained Large Language Model, an emulation of a front-end environment, and finally logging and data collection mechanisms.

2.2 AI Enabled Honeypot

The proposed system would operate as an adaptive honeypot that is capable of producing realistic and contextually correct responses to an attacker's activity. The attacker would connect to the honeypot through a simulated SSH interface, after connecting all their inputs into the honeypot would be captured by a command handling layer. This layer would serve to interpret the command and determine if it should return a predefined template response or if the input should be sent to the AI response engine in the background. This AI response engine would be built around an LLM and be used for generating realistic system outputs based on the input that has been passed back to it. The engine should model its output based on the current session context, such as the attackers' previous actions, their location in the system, and the directory structure created for the framework.

The system will employ a hybrid approach. Where in common commands that would not typically expose the honeypot such as ls, cat or whoami would use a static template in the framework, uncommon or unexpected inputs would trigger the command handling layer to send the input to the AI Response Engine, where the LLM would then produce a response. The AI Response Engine will be guided by strict guardrails to ensure that only a terminal style output is generated. Safeguards will be put in place to ensure that the model never attempts to execute or forward the inputs to a real operating system.

A lightweight simulated operating system state is to be maintained in a local SQLite database. This database will store the virtual file structure, user accounts, and the process list that is presented to attackers. Session activity will also be logged in this database; this will include items such as the commands the attacker provided and the responses that were generated and provided back.

The project will also include the deployment of the Cowrie honeypot framework. This deployment will be used provide a baseline for the testing being conducted against the AI

Enabled Honeypot. Both the Cowrie Honeypot and the AI Enabled Honeypot will be deployed in isolated virtual machines. This is to prevent and external exposure and provide an isolated and safe testing environment. Python will be utilised as the main scripting language to take advantage of its numerous libraries (i.e. AsyncSSH for SSH emulation). The AI component will use a managed API model.

The resulting system will allow for controlled testing where we will be able to simulate attacker actions against the adaptive model implementation and the static model implementation and compare the resulting data, such as session duration and unique commands used.

2.3 User Groups

The AI Enabled Honeypot is intended for the same user groups as a Traditional Honeypot, this includes researchers, security analysts, and educators.
For researchers it would provide a controlled environment to study attacker behaviour. The AI Enabled Honeypot should extend attacker engagement providing an increase to the quality and quantity of behavioural data made available for review.
For Security Operations Centre (SOCs) teams the honeypot would serve as an effective distraction from their live system. The extended engagement that the AI Enabled Honeypot provides would not only provide SOC teams behavioural data same as with researchers but also keep attackers engaged longer than a static honeypot.
Educators can use the system as a demonstration tool in a cybersecurity course, showing students realistic attack scenarios without risk.

3.0 Deliverables

The deliverables for this project are divided into two categories: core deliverables, and secondary deliverables. Core deliverables are deliverables deemed essential for the functionality of the AI Enabled Honeypot. Secondary deliverables are deliverables that serve to support the core deliverables and improve the project's usability.

3.1 Core Deliverables

3.1.1 System-Specific Core Deliverables

System-Specific Core Deliverables are deliverables that directly contribute to the development of the system these will include the following:

AI Driven SSH Honeypot Prototype

This project will focus on creating a SSH Honeypot with AI driven responses to attacker commands. This honeypot will emulate an SSH accessible command shell that the attacker can interact with. The attackers' commands will be passed to a trained LLM in the background to generate responses. The LLM integration will serve to resolve the bottle neck of needing to predefine every response to attackers' commands. It will allow for a greater coverage of commands and should keep the attacker engaged for longer by giving the impression that they have more control in the system than they do.

Baseline Static Honeypot

A static honeypot will need to be utilised to allow for comparisons between the static and adaptive honeypots to be made. This static honeypot will either be a basic install and configuration of the Cowrie Honeypot software or alternatively an implementation of the honeypot framework that the adaptive honeypot is based on but without the LLM behind it. Both implementations have pros and cons.

AI Response Engine (Large Language Model Integration)

The AI Response Engine will form the core of this project. A Large Language Model will be trained to develop realistic responses to an attackers' commands. The Mitre Att&CK (Mitre Attack) Framework will be utilised to train the LLM on how to respond to common attack methods. Input to the LLM will be sanitised before being sent to ensure that 1. The LLM is not discovered by the attacker 2. To ensure that if discovered the attacker cannot take advantage of the LLM in any way. There are several options for the LLM, such as: OpenAI GPT Models, Llama 3, Gemini.

3.1.1 Non-System-Specific Core Deliverables

Non-System-Specific Core Deliverables are deliverables that do not contribute to the development of the system but are required to fulfil the project.

Research Report

This Project will include a research report that will document the planned development process of the project, the potential technologies that will be used such as Cowrie, Ubuntu, and Llama 3, testing methodologies and early research findings. The report will serve as a complete record of technical and research areas for the project. The report should provide a clear and concise understanding to any that read it of how the idea for the project was envisioned, how the system will be developed and how it will be evaluated.

Project Report

The project report will present a full account of the project's development and outcomes. It will include Technical Specifications, Installation and deployment guidelines, test findings and recommendations on how the project could be further built upon.

Showcase Website

The Project will also include a website to highlight the project’s documentation and its milestones. The website will allow for any interested user to gain more information on the project, high level overview, features, test cases, etc.

3.2 Secondary Deliverables

Preset Honeypot Environments

A set of predefined system environments that could be loaded based on the user’s requirements. These environments would emulate different types of organisations, such as financial enterprises, educational institutions, or personal servers. Each preset would have defining characteristics through differences in their file structures, user accounts, and the simulated data made available to the attacker. This would add to the usability of the system by allowing users to easily select different honeypot profiles.

Auto Generated Reports

After an attacker has interacted with the honeypot, users will have the option to automatically generate a summary report. The large language model (LLM) will analyse the system logs and create a high-level overview report of the attacker’s actions for the user. This report is intended to make the post-engagement review more efficient by highlighting key actions and potential objectives observed during the session. The full log data will remain available.

4.0 Functionalities

Functionalities are specific actions/capabilities that the system must be able to complete to meet both user expectations and project requirements. Functionalities are further divided into Core System, Additional, Non-Functional and Out of Scope.

4.1 Core Functionalities

Attacker Session Handling

The honeypot will accept connections over SSH and present a realistic login banner and an emulated shell to mimic a genuine server. Attackers inputted commands and keystrokes will be saved. It will reproduce basic TTY behaviour (backspace, line endings, etc).

Command Parsing and Routing

Attacker sent commands are to be parsed and subsequently routed. This will be based on if the command is a “common/basic” command or a more “complex/in depth command.” Basic commands will be handled by local templates. In depth/more complex commands will be router to the AI response engine (LLM), and the LLM will be used to create on demand simulated response that mimic a realistic output.

AI Response Generation

When commands are sent to the AI response engine, the system will build a compact session context profile. This profile will include items such as the attacker’s previous commands, prompt details, preset command outputs (to provided initial/ baseline system context). This profile is passed to the LLM which then generates a realistic CLI styled output. The LLM will provide outputs that give the impression of a much larger system being in place (underlying filesystem in place, user accounts, process snapshots, etc). Reponses will be sanitised and validated to ensure non-terminal like text is not outputted. If the LLM fails or times out a generic error will be outputted.

System Emulation

The honeypot will provide a basic emulated shell, designed only to accept input and display output. It will not maintain a persistent or functional system. The LLM and pre-defined templates will be relied upon to provide outputs mimicking a real system.

Logging and Data Collection

All session activity will be logged for analysis. This will include commands sent, responses generated, timestamps, and LLM latency.

4.2 Additional Functionalities

Preset Honeypot Environments

Users will be able to choose from a selection of predefined system environment presets. Each preset will initialise the honeypot with a separate set of predefined system responses to basic commands. When a preset is selected the AI response engine is configured to simulate the selected environment when creating responses.

Auto-Generated Reports

Users will be able to request a session summary report after the attacker has exited the honeypot. The LLM will analyse the session logs and generate a high-level overview. This report will include the attacker’s actions and potential objectives.

4.3 Non-Functional Requirements

Believability and Consistency

Generated responses must appear authentic and remain consistent to the environment throughout an attacker's session. Methods such as variations in timestamps, process IDs, latency, and contextual prompts will be used to enhance realism and prevent fingerprinting.

Reliability

The system will ensure that a response is always returned by using timeouts and general error outputs as a fallback mechanism. If the LLM times out or fails unexpectedly the system will output a general system error and revert to the pre-defined template outputs.

Performance

The system will be optimised for low latency. However, to maintain believability in the system, template responses will have their latency artificially increased to match the delay in the AI generated outputs. Response times will ideally be no more than a second. Efforts will be made to have the system handle multiple concurrent sessions.

Maintainability

The system architecture will separate the AI response engine, SSH framework, and supporting modules. This will allow for easier updates, improvements, and maintenance.

Security and Containment

For the purpose of this project the honeypot will remain isolated in a secure test environment and will never be exposed to external networks. Guardrails will be put in place to ensure attackers cannot interact directly with the LLM. Payloads provided by the attacker will not be executed or stored.

4.4 Out of Scope

This prototype system will not:

- Be deployed on a public network or on live infrastructure.
- Store, execute, or analyse malware samples.
- Attempt to emulate a full operating system beyond CLI emulation.

5.0 System Architecture and Technologies

5.1 System Architecture

The AI Enabled Honeypot follows a modular, networked architecture. The external facing SSH frontend (note: for the purpose of this project, it will remain internally hosted and not publicly assessable) accepts a connection and presents an attacker with an emulated CLI. When the attacker issues a command, the emulated shell will either produce a predefined template output if one exists for the given command or forward the command to the AI response engine to generate an appropriate output. All outputs are returned to the emulated shell; every interaction and output is logged for user analysis. The diagram below showcases the flow of data through the system.

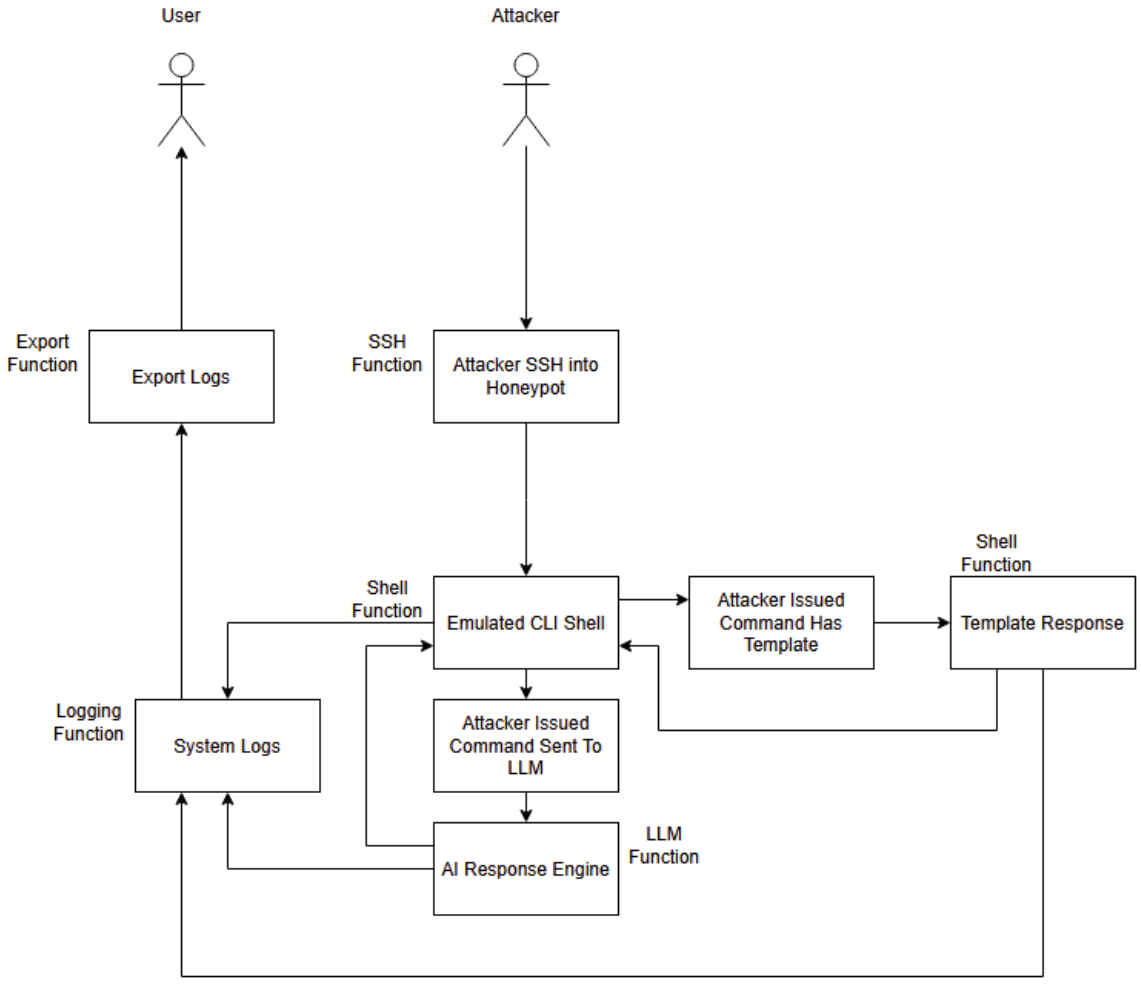

Figure 1 System Architecture Diagram

Data Flow

1. Attacker connects to the honeypot via the SSH protocol.
2. Once connected the attacker is placed in an emulated CLI shell.
3. When the attacker issues a command, the system will first check to see if a predefined template response exists. If found the template's output is returned to the CLI.
4. If no template exists for the issued command, then the command is sent to the AI Response Engine (after being sanitised).
5. The AI Response Engine leverages its LLM to generate an appropriate and believable response, and forwards that response back to the CLI.
6. All commands entered and responses produced (LLM and Template) are logged for later analysis.
7. The user can then analyse these logs after the session has ended.

5.2 Technologies

The development of the AI Enabled Honeypot will utilise several technologies to support its development, improve its functionality, and increase its reliability.

Python

Python will serve as the project's primary programming language. Python was chosen due to its wide selection of libraries and personal familiarity with the language. Python will be used to build the SSH honeypot, the AI Response Engine, and the logging module. Additional features/deliverables will also be programmed in Python.

Paramiko Library

The Paramiko library will be utilised to create and implement the SSH functionality of the honeypot, allowing attackers to connect and interact with the emulated CLI shell.

Llama 3

Llama 3 will serve as the Large Language Model powering the AI Response Engine. Llama 3 was chosen for its ability to be run locally; this allows for full offline operations. This allows the honeypot to remain completely isolated from external networks during testing. In a theoretical production environment only the SSH interface would be externally exposed.

Pandas

Pandas will be used for the analysis of the collected log files. It provides tools for filtering, aggregating, and visualising session data such as command frequency, session length, dwell time, and incoming IP addresses.

SQLite

SQLite will be used to provide local storage for session logs and configuration presets (secondary deliverable). SQLite offers simple integration with Python and requires no dedicated database server.

6.0 Use Cases

Use cases demonstrate how the AI Enabled Honeypot is intended to be used by users. Different types of users of the system will interact with different system functions. For example, a security analyst would interact with the system via deployment and gathering of system logs. An attacker on the other hand is also a user of the system; this user would interact with the honeypot via the emulated CLI interface and the AI Response Engine through the commands they enter.

6.1 Detailed Use Cases

This section provides detailed descriptions of the various use cases identified with using the AI Enabled Honeypot. Each use case can be broken down into its components: Use Case Name, Description, Actor(s), Pre-Conditions, Post-Conditions, Trigger, Basic Flow, Alternative Flow.

Use Case 1	Descriptions
Use Case Name	Connection to Honeypot
Description	Attacker connects to the honeypot server via the SSH protocol with the honeypot's IP address.
Actor(s)	Attacker
Pre-Conditions	Attacker must have obtained the honeypot's IP address, Threat Actor must have knowledge on how to use the SSH protocol.
Post-Conditions	Attacker will successfully connect to the honeypot and be presented with the honeypot's emulated CLI shell.
Trigger	Attacker attempts to connect to the honeypot via SSH.
Basic Flow	1. Attacker obtains honeypot IP address. 2. Attacker attempts to connect to the honeypot's server via SSH. 3. If successful, the attacker will be presented with the honeypot's emulated CLI shell. 4. System logs relevant connection information, such as Source IP and connection time.
Alternate Flow	1. Attacker fails to connect through a failure in using the SSH protocol, incorrect IP address being provided or incorrect credentials being passed, resulting in an SSH error.

Use Case 2	Descriptions
Use Case Name	Sending Commands through the CLI
Description	Once connected to the honeypot the attacker begins to input commands into the emulated CLI shell. The honeypot will then check to see if the command matches a predefined template or if it needs to be passed to the AI Response Generator. The system will also log this command.
Actor(s)	Attacker, System
Pre-Conditions	Attacker is connected to the honeypot.
Post-Conditions	Attacker's issued command is logged. The command is sent to the AI Response Engine if needed.
Trigger	Attacker issues a command to the emulated CLI.
Basic Flow	1. Attacker issues a command to the honeypot. 2. The honeypot system checks to see if the issued command has a template response in the system. 3. If not, the command is forwarded to the AI Response Generator. 4. Command is logged by the system.
Alternate Flow	1. If the command does not have a templated response and the system fails to forward the command a general error will be outputted to the screen.

Use Case 3	Descriptions
Use Case Name	System Template Output
Description	If a command is issued to the honeypot system that matches one of the pre-set templates present, then the corresponding output will be presented.
Actor(s)	System
Pre-Conditions	A command that matches a pre-set template has been issued to the emulated CLI shell.
Post-Conditions	The template's output is outputted to the CLI.
Trigger	A command is sent through the CLI that matches a template.
Basic Flow	1. Command is issued to the system. 2. System checks to see if the command matches any of the preset response templates. 3. If so, then that templates response is outputted. 4. Command and output provided are recorded.
Alternate Flow	1. The command does not match any pre-set templates; command is sent to AI Response Engine. 2. Technical error occurs, catch all error message is outputted.

Use Case 4	Descriptions
Use Case Name	AI Response Engine Output
Description	If a command is issued to the honeypot that does not match any of the preset template presets, then the command is forwarded to the AI Response Engine module so an output can be generated.
Actor(s)	System, AI Response Engine
Pre-Conditions	A command that matches a pre-set template has been issued to the emulated CLI shell.
Post-Conditions	The template's output is outputted to the CLI.
Trigger	A command is sent through the CLI that matches a template.
Basic Flow	1. Command is issued to the system. 2. System checks to see if the command matches any of the preset response templates. 3. The issued command does not match any of the response templates. 4. The command is forwarded to the AI Response Engine. 5. The AI Response Engine generates a realistic and contextually correct output for the command. 6. The generated output is returned is outputted. 7. Command and output provided are recorded.
Alternate Flow	1. The command matches any of the pre-set templates; a template response is returned. 2. Technical error occurs, catch all error message is outputted.

Use Case 5	Descriptions
Use Case Name	Export Logs
Description	Export all log data collected by the system.
Actor(s)	User, System
Pre-Conditions	Logs have been generated from an attacker's session with the honeypot.
Post-Conditions	Log files are downloaded to the user's machine.
Trigger	User hits the "Export Logs" button.
Basic Flow	1. User clicks the "Export Logs" button. 2. The system retrieves the stored log files for the previous session. 3. Files are downloaded to the user's machine.
Alternate Flow	Logs fail to download; error message is displayed.

Use Case 6	Descriptions
Use Case Name	Deploy System
Description	Configuration and deployment of the AI Enabled Honeypot on an externally facing server.
Actor(s)	User
Pre-Conditions	Organisation’s server must be stood up and exposed to external networks.
Post-Conditions	The AI Enabled Honeypot is deployed and working on a server.
Trigger	User begins configuration and deployment of system.
Basic Flow	1. User stands up an externally facing server. 2. User hosts the AI Response Engine on the server. 3. User deploys honeypot software to the server. 4. User tests honeypot before opening connections to ensure system is secure and airtight. 5. System is configured with purposefully weak username and password combination to allow for increased attacker engagement. 6. User opens port 22 to allow for SSH connections.
Alternate Flow	If deployment fails, user troubleshoots deployment issues.

7.0 Project Plan

This section outlines the project's development plan, it includes a development timeline, progress goals, and core tasks. It also covers the development and testing of the AI Enabled Honeypot. The plan will be divided into three phases: research and preparation, development, and testing. The project will rely on each phase being completed one after the other, with each building upon the previous phase.

7.1 Project Phases

1: Research and Preparation

This phase will focus on conducting the necessary research for the project. This will help refine the projects scope and reaffirm the project requirements. These findings will assist in ensuring the following development and testing phases progress smoothly.

Research Report

A completed research report will be written for this project. This document will provide greater insights into multiple aspects of the project, including the evaluation of external tools and technologies that could potentially be used, assisting with the decision of which Large Language Model to utilise, existing honeypot frameworks that are available to users, their strengths and weaknesses, libraries that could be utilised, and previous attempts to develop a system similar to this. The research report will aim to evaluate what is feasible within the projects scope. It will help ensure the suitability of all tools and technologies selected and the feasibility of the project, learning from previous endeavours in this space.

Tool Testing and Preparation

During this stage the various tools and technologies researched will be tested to evaluate their functionality and to determine which are the most suitable for the project. Existing honeypot frameworks will be tested to identify their strengths and weaknesses. These frameworks will also be tested to determine which framework would be the most suitable for comparison testing against the AI Enabled Honeypot. Additional testing will also be conducted to determine the possibility of modifying an existing honeypot framework to work with the AI Response Engine. Also in this phase, preparations for development will be completed. This will include the establishment of an effective development environment and the standup of needed infrastructure such as isolated virtual machines.

2: Development

The system's development will be broken down into individual module development: Honeypot Framework, AI Response Engine, and Integration. Each module will be developed and tested separately before they are fully integrated with each other.

Honeypot Framework

The honeypot framework module will serve as the foundation of the AI Enabled Honeypot. It will provide the core infrastructure for handling the attacker interactions and emulation of the CLI Shell. The framework will include the SSH functions, shell emulation functions and the template response functions. The framework will be developed using Python and libraries such as the Paramiko library for SSH implementation will be utilised. The shell emulation function will first focus on replicating a simple terminal experience that allows attackers to execute basic commands. As development progresses this will be narrowed down to replicate a server that provides a specific service (i.e. web server). This will allow for easier training of the AI Response Engine as it will only need to be trained to produce responses related to the chosen service. The response templates will also be pre-selected and included directly in the module. If time allows these will be moved to their own module to allow for more varied deployment of the system. This module will also include the command handling and system logging functions. The Honeypot Framework will act as the backbone of the project; it will serve as a platform that the AI Response Engine can be later integrated into.

AI Response Engine

After the base honeypot is established, the next step will be the development of the AI Response Engine. The AI Response Engine will be responsible for generating realistic and context aware outputs to any command that does not match a predefined template within the system. The core of the AI Response Engine will be the Large Language Model (LLM) responsible for generating the adaptive responses. The engine will take input sent through the emulated CLI, construct a prompt that includes any relevant session information (recent commands, environment set up) and send this to the LLM. The LLM will then generate a believable response to the issued command. The output will be validated to ensure that no information about AI Response Engine is passed back. The output is then returned to the CLI and presented to the attacker. The AI Response Engine will be written in Python. Safety mechanisms will be built into the engine to prevent direct interaction with the LLM, enforce time limits, and handle invalid outputs. If an error occurs, the framework will produce a generic error.

Integration

Finally, the AI Response Engine will be linked with the Honeypot Framework. This will allow the honeypot to extend attacker engagement via adaptive outputs and prevent fingerprinting of the system.

3: Testing

This is the final phase of the project plan; it will involve extensive testing to ensure the system is functional and usable. It will also be tested to see how it compares to a static honeypot system. Testing will be broken down into the following categories:
Functional Testing, Usability Testing, Performance Testing, and Comparison Testing.

Functional Testing
Test the core features of the system to ensure full functionality. This will include testing the template responses, response generation, and logging of session activity.

Usability Testing
Evaluate how the attacker may navigate the system and how they connect to the system. Also, how a user can download the system logs and generate a report from the LLM.

Performance Testing
Ensure that the AI Response Engine can generate responses in a reasonable time frame. Test how the system can handle multiple connections at the one time.

Comparison Testing
Compare the AI Enabled Honeypot to a Static Honeypot Framework. Test the two against a simulated attacker, measure how long the attacker interacts with each system, how each system generates responses, and how long it takes for the system to be identified as a honeypot.

7.2 Timeline

The timeline shows major project milestones that will need to be reached. Each milestone has an estimated start and end date. Dates are subject to change as the project progresses. Some milestones may prove more challenging to complete then initially estimated. Some milestones may overlap with each other.

Figure 2 Project Timeline Table

Milestone	Start	End	Duration (Days)
Research and Preparation	**27/10/25**	**5/12/25**	**40**
Research Report	27/10/25	5/12/25	40
Development	**5/12/25**	**4/03/26**	**90**
Honeypot Framework	5/12/25	4/01/26	31
AI Response Engine	4/01/26	31/01/26	28
Integration	31/01/26	6/02/26	7
Initial Testing	6/02/26	10/02/26	5
Static Honeypot	10/02/26	14/02/26	5
Secondary Deliverables	**14/02/26**	**4/03/26**	**19**
Report Generation	14/02/26	23/02/26	10
Preset Environments	23/02/26	4/03/26	10
Testing	**4/03/26**	**16/03/26**	**13**
Functional Testing	4/03/26	10/03/26	7
Usability Testing	4/03/26	10/03/26	7
Performance Testing	10/03/26	16/03/26	7
Comparison Testing	10/03/26	16/03/26	7
Conclusion	**16/03/26**	**31/03/26**	**18**
Project Report	16/03/26	29/03/26	14
Showcase Website	16/03/26	29/03/26	14

Final Cleanup	29/03/26	31/03/26	3

8.0 References

- 0xdf (2023). *Creating a SSH Honeypot with Python*. [online] YouTube. Available at: https://www.youtube.com/watch?v=HO1h57CiF98 [Accessed 14 Oct. 2025].

- Cowrie (2020). *cowrie/cowrie*. [online] GitHub. Available at: https://github.com/cowrie/cowrie [Accessed 13 Oct. 2025].

- David Bombal (2025). *Brute Force SSH & Build a Honeypot Now (Hydra and Cowrie Demo)*. [online] YouTube. Available at: https://www.youtube.com/watch?v=uSohtNwQXuI [Accessed 13 Oct. 2025].

- Grant Collins (2024). *ssh_honeypy/ssh_honeypot.py at main · collinsmc23/ssh_honeypy*. [online] GitHub. Available at: https://github.com/collinsmc23/ssh_honeypy/blob/main/ssh_honeypot.py [Accessed 14 Oct. 2025].

- Matt Adams (2023). *GitHub - mrwadams/honeyagents: HoneyAgents is a PoC demo of an AI-driven system that combines honeypots with autonomous AI agents to detect and mitigate cyber threats. Features include intelligent threat analysis, automated deny list updates, and detailed natural language threat reports.* [online] GitHub. Available at: https://github.com/mrwadams/honeyagents?tab=readme-ov-file [Accessed 13 Oct. 2025].

Appendix

Images