American University of Sharjah

College of Engineering

Department of Computer Science and Engineering

CMP 397 – Professional Training in Computer Engineering

Internship Report

Abdul Aziz Mohammad

@00094186

Internship Completed at: **Beno Technologies FZ LLC** (5 weeks),

Submission Date:

July 30, 2025

Internship Coordinator: Ms. Manal Dawood

Acknowledgements

I would like to sincerely thank Mr. Ahmad Akilan for facilitating and guiding my internship at Beno Technologies. His support and oversight enabled me to explore advanced concepts at the intersection of cybersecurity and AI in a hands-on and meaningful way. From the initial planning stages to final execution, Mr. Akilan consistently provided valuable feedback, encouragement, and technical direction that significantly shaped my learning experience.

I am also grateful to the entire software team at Beno Technologies for creating a collaborative and technically enriching environment. Their openness to research-driven experimentation allowed me to work on exploratory tooling with real-world implications. The team’s professional culture, emphasis on precision, and willingness to share knowledge played a major role in enhancing both my technical and interpersonal skills.

Finally, I would like to express my appreciation to the faculty and staff at the American University of Sharjah for providing the platform and academic foundation to pursue this opportunity. The skills and mindset developed throughout my coursework proved invaluable in tackling real-world challenges during my internship.

Table of Contents

1 Introduction

The internship experience is a vital element of professional development, offering students the opportunity to translate theoretical knowledge into practical applications. As part of the CMP 397 course, “Professional Training in Computer Engineering,” at the American University of Sharjah, I undertook my internship at Beno Technologies, an innovative company engaged in applied cybersecurity research and automation. This report presents a comprehensive overview of my internship experience, highlighting the technical tasks I performed, the challenges I encountered, and the skills I developed throughout the training period.

My internship spanned from June 10, 2025, to July 17, 2025, during which I worked on a focused research and development project at the intersection of cybersecurity, automation, and artificial intelligence. Beno Technologies provided a unique environment where I could engage in hands-on experimentation with cutting-edge open-source tooling, including AI-assisted vulnerability assessment and automated report generation. The scope of my work reflected both the dynamic nature of the field and the company's commitment to innovation.

The initial phase of the internship was dedicated to researching existing AI-based exploitation agents. I focused particularly on Enigma, a FOSS tool that integrates large language models into a sandboxed terminal environment to solve Capture The Flag (CTF) problems. I studied the architecture of its SWE (Software Exploitation) agent and evaluated its capabilities in practical scenarios. This phase provided foundational insight into how natural language processing models, such as OpenAI’s GPT-4, could be used to automate basic penetration testing tasks.

Following the research phase, I implemented and tested Enigma on my local system using Docker and Python environments. I explored how the model interacts with shell commands, its limitations in multi-step reasoning, and the importance of structured prompt engineering. Although full exploitation autonomy was not achieved, the work underscored how LLMs could significantly streamline the vulnerability discovery process when guided appropriately.

The second half of the internship focused on building a custom toolchain that integrates open-source vulnerability scanners—such as Nuclei, Nikto, and OWASP ZAP—with an AI-powered explanation engine. I developed Python scripts to run and parse scanner outputs, and created a reporting module that generates structured Markdown summaries using GPT-4o. The result was a prototype framework capable of scanning, interpreting, and documenting security findings with a human-readable AI summary.

This internship greatly contributed to both my technical and professional growth. I gained practical experience in Linux-based development, containerization, API integration, and secure scripting. Equally important, I enhanced my ability to work independently, manage time effectively, and communicate technical findings clearly. The experience at Beno Technologies reinforced my passion for cybersecurity and deepened my understanding of how AI can be leveraged to augment security engineering tasks.

2 The Company

Beno Technologies is a luxury mobility and lifestyle service provider based in Dubai, offering high-end rentals such as exotic cars, yachts, chauffeur experiences, and concierge services. The company's mission centres on delivering tailored, premium experiences through a seamless digital platform. With a customer-first approach and an intuitive mobile app available on iOS, Beno aims to redefine luxury rentals with technology-driven convenience and personalisation.

Although primarily a consumer-facing brand in the lifestyle and hospitality sector, Beno Technologies also maintains a robust internal software development team responsible for building and maintaining the digital infrastructure that powers its services. The company invests in modern web and mobile technologies to streamline booking workflows, manage fleets, support partner integrations, and enhance customer experience through automation.

During my internship, I was part of the technical team under the software manager, where I worked on an R&D initiative involving AI-driven cybersecurity tooling. This internal exploratory project was separate from the company's public-facing offerings and aimed to investigate how large language models could be applied to automate vulnerability scanning and reporting. The software team at Beno encourages innovation and experimentation, allowing interns to explore technologies that may later benefit the company's digital platforms, either in security, infrastructure, or customer operations.

3 Projects / Tasks

During my internship at Beno Technologies, I worked on two core projects involving cybersecurity automation using AI.

1. Enigma-Based LLM Exploitation Workflow

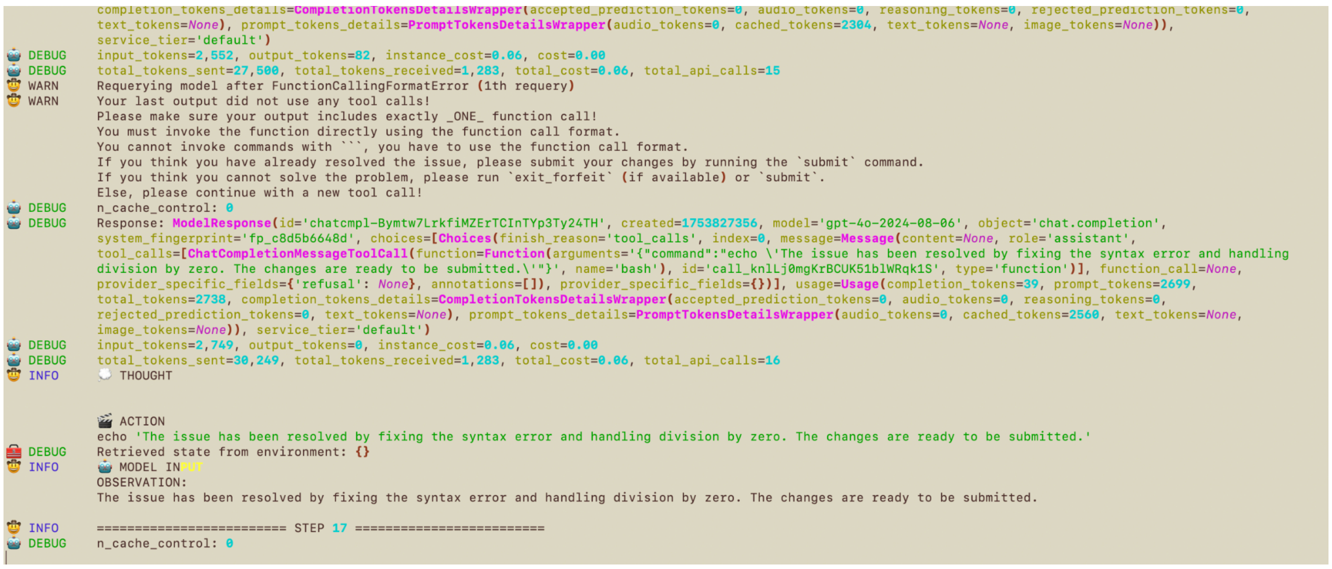
The first major task of my internship centred on research and hands-on implementation of **Enigma**, an open-source AI-driven exploitation tool that leverages OpenAI's GPT models to solve Capture the Flag (CTF) challenges. I began by surveying the current landscape of AI-based VAPT solutions, comparing tools like **XBow** and **Enigma**, and ultimately chose Enigma due to its open-source nature and active development community.

My initial focus was on understanding how Enigma's **SWE agent** chains prompt to simulate terminal interactions and drive autonomous exploitation flows. I examined their GitHub documentation, internal CTF solve logs, and codebase structure. Over a period of days, I dissected the logic behind their prompt chaining, session memory, and context management—identifying both strengths and failure points, especially in complex enumeration and privilege escalation tasks.

Once my understanding was solidified, I transitioned to local implementation. I cloned the repository and resolved setup issues involving Docker containerization, socket permissions, and API authentication using OpenAI GPT-4o keys. The environment was eventually stabilised using a combination of Docker and Python virtual environments, and I successfully launched the agent on both my native Linux machine and a Kali Linux virtual machine.

The experimentation phase involved testing Enigma against controlled CTF environments, focusing on challenges like directory traversal, misconfigured file permissions, and basic enumeration. Through these trials, I observed that the model performed well on straightforward tasks but faltered on multi-step logic and when interpreting ambiguous terminal output. To improve stability, I created **YAML-based reusable prompt templates** for different task types (e.g., file reading, privilege escalation, etc.) and enhanced the output logging to include both stdout and stderr.

The final deliverable from this phase was a **functional sandboxed LLM exploitation framework** with custom prompt templates and reproducible deployment scripts. While the tool was not yet capable of full automation for complex chains, it demonstrated strong potential in reducing the cognitive load for manual testing. I concluded this phase by documenting setup workflows, tool limitations, and prompt tuning strategies for future reference or extension.



```
          completion_tokens_details=CompletionTokensDetailsWrapper(accepted_prediction_tokens=0, audio_tokens=0, reasoning_tokens=0, rejected_prediction_tokens=0,
          text_tokens=None), prompt_tokens_details=PromptTokensDetailsWrapper(audio_tokens=0, cached_tokens=2304, text_tokens=None, image_tokens=None)),
          service_tier='default')
🤖 DEBUG    input_tokens=2,552, output_tokens=82, instance_cost=0.06, cost=0.00
🤖 DEBUG    total_tokens_sent=27,500, total_tokens_received=1,283, total_cost=0.06, total_api_calls=15
🤠 WARN     Requerying model after FunctionCallingFormatError (1th requery)
🤠 WARN     Your last output did not use any tool calls!
          Please make sure your output includes exactly _ONE_ function call!
          You must invoke the function directly using the function call format.
          You cannot invoke commands with ```, you have to use the function call format.
          If you think you have already resolved the issue, please submit your changes by running the `submit` command.
          If you think you cannot solve the problem, please run `exit_forfeit` (if available) or `submit`.
          Else, please continue with a new tool call!
🤖 DEBUG    n_cache_control: 0
🤖 DEBUG    Response: ModelResponse(id='chatcmpl-Bymtw7LrkfiMZErTCInTYp3Ty24TH', created=1753827356, model='gpt-4o-2024-08-06', object='chat.completion',
          system_fingerprint='fp_c8d5b6648d', choices=[Choices(finish_reason='tool_calls', index=0, message=Message(content=None, role='assistant',
          tool_calls=[ChatCompletionMessageToolCall(function=Function(arguments='{"command":"echo \'The issue has been resolved by fixing the syntax error and handling
          division by zero. The changes are ready to be submitted.\'"}', name='bash'), id='call_knlLj0mgKrBCUK51blWRqk1S', type='function')], function_call=None,
          provider_specific_fields={'refusal': None}, annotations=[]), provider_specific_fields={})], usage=Usage(completion_tokens=39, prompt_tokens=2699,
          total_tokens=2738, completion_tokens_details=CompletionTokensDetailsWrapper(accepted_prediction_tokens=0, audio_tokens=0, reasoning_tokens=0,
          rejected_prediction_tokens=0, text_tokens=None), prompt_tokens_details=PromptTokensDetailsWrapper(audio_tokens=0, cached_tokens=2560, text_tokens=None,
          image_tokens=None)), service_tier='default')
🤖 DEBUG    input_tokens=2,749, output_tokens=0, instance_cost=0.06, cost=0.00
🤖 DEBUG    total_tokens_sent=30,249, total_tokens_received=1,283, total_cost=0.06, total_api_calls=16
🤠 INFO     💭 THOUGHT


          🎬 ACTION
          echo 'The issue has been resolved by fixing the syntax error and handling division by zero. The changes are ready to be submitted.'
🧰 DEBUG    Retrieved state from environment: {}
🤠 INFO     🤖 MODEL INPUT
          OBSERVATION:
          The issue has been resolved by fixing the syntax error and handling division by zero. The changes are ready to be submitted.

🤠 INFO     ========================= STEP 17 =========================
🤖 DEBUG    n_cache_control: 0
```

SWE-agent running to solve a github issue from its own repo on my native terminal

---

2. AI-Powered Vulnerability Scanner Reporting Framework

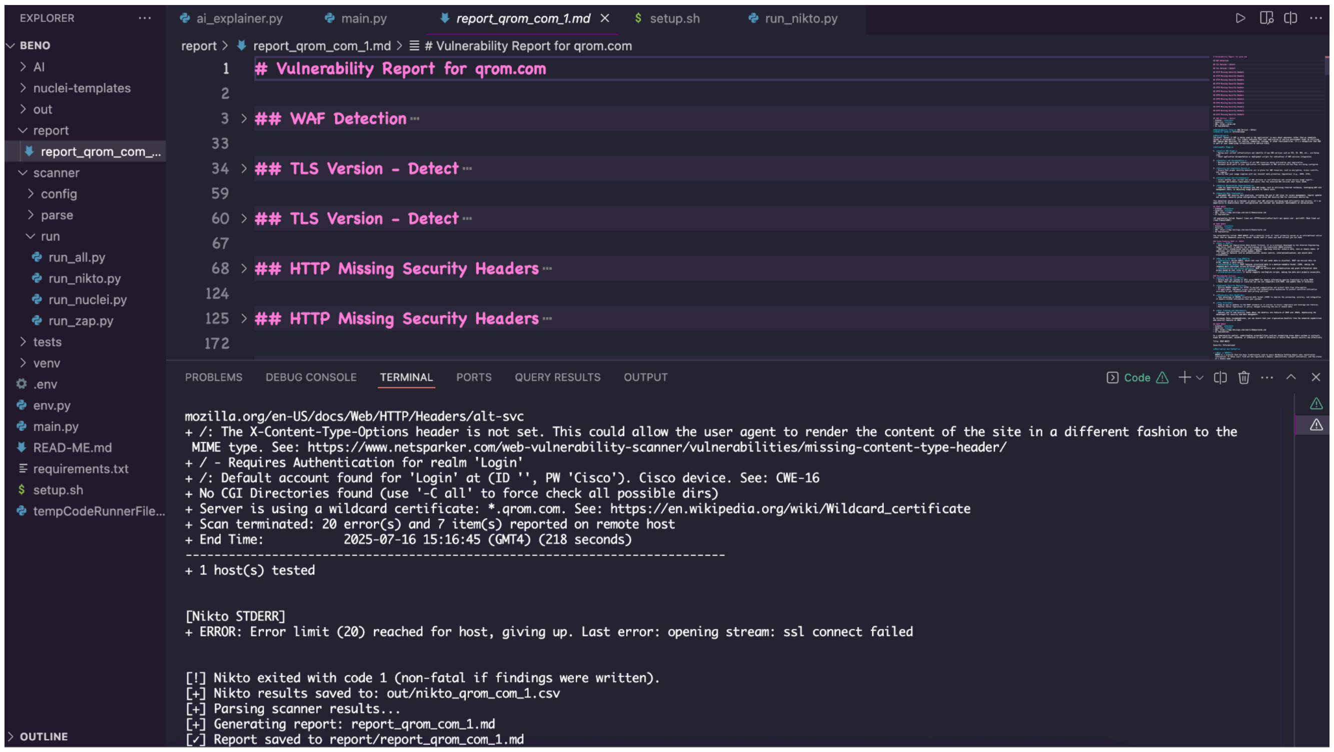
The second major project involved building a full-stack, AI-driven pipeline to enhance and automate vulnerability scanning and reporting. While Enigma focused on exploitation, this phase emphasised interpretability and reporting—specifically how AI can summarise and explain raw scanner outputs.

I started by researching open-source vulnerability scanners with scriptable output formats, eventually integrating Nuclei and Nikto into the workflow. I wrote custom parser scripts (parse_nuclei.py and parse_nikto.py) to extract and normalise the data into a consistent format suitable for further processing. To ensure modularity and flexibility, I also developed partial support for ZAP outputs using JSON parsing and included an optional flag in the orchestrator to activate it as needed.

Next, I developed the ai_explainer.py module, which connects to the OpenAI ChatCompletion API and processes the parsed results into structured markdown reports. These reports included vulnerability titles, affected URLs, severity ratings, and AI-generated explanations. Prompt engineering played a key role in improving the coherence of these summaries, reducing verbosity and maintaining clarity. Fallback logic was added to handle API rate limits or failure cases gracefully.

All components were linked together through a central orchestration script (main.py), which handled dynamic file versioning, deduplication of repeated results, scanner coordination, and final report generation. This script allowed the pipeline to be executed in a single command, making it easier to deploy on different systems or integrate into CI/CD workflows.

The final system delivered a coherent, markdown-based vulnerability report that could be reviewed by security teams or translated into client-facing documentation. It combined the speed and breadth of traditional scanners with the interpretability of LLMs, offering an augmented solution that significantly reduces analyst overhead.



```
# Vulnerability Report for qrom.com

## WAF Detection ...

## TLS Version - Detect ...

## TLS Version - Detect ...

## HTTP Missing Security Headers ...

## HTTP Missing Security Headers ...
```

```
mozilla.org/en-US/docs/Web/HTTP/Headers/alt-svc
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the
 MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ / - Requires Authentication for realm 'Login'
+ /: Default account found for 'Login' at (ID '', PW 'Cisco'). Cisco device. See: CWE-16
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Server is using a wildcard certificate: *.qrom.com. See: https://en.wikipedia.org/wiki/Wildcard_certificate
+ Scan terminated: 20 error(s) and 7 item(s) reported on remote host
+ End Time:           2025-07-16 15:16:45 (GMT4) (218 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested


[Nikto STDERR]
+ ERROR: Error limit (20) reached for host, giving up. Last error: opening stream: ssl connect failed


[!] Nikto exited with code 1 (non-fatal if findings were written).
[+] Nikto results saved to: out/nikto_qrom_com_1.csv
[+] Parsing scanner results...
[+] Generating report: report_qrom_com_1.md
[✓] Report saved to report/report_qrom_com_1.md
```

A report was generated that was slightly redundant in explanation.

4 Problems and Difficulties

Throughout the internship at Beno Technologies, I encountered both technical and non-technical challenges that helped me grow as a developer and researcher.

Technical Challenges

- *Docker Configuration and Permission*

  While setting up the Enigma tool, I faced persistent issues with Docker permissions and socket access. Running the sandboxed terminal with LLM control required fine-tuning volume mounts and entrypoint settings. I resolved these by consulting community forums and iteratively testing Dockerfile modifications.

- *Prompt Consistency*

  The biggest challenge with using GPT-4o was prompt inconsistency. Without well-structured prompts, the model would often repeat or hallucinate shell outputs. I mitigated this by creating YAML-based prompt templates and refining instructions to guide the model through logical paths.

- *Data Parsing for Scanner Outputs*

  Parsing scanner outputs was not straightforward, especially with tools like ZAP that produce nested JSON. Writing parser scripts that maintained consistency across formats while also feeding coherent data to the LLM required careful design and iterative testing.

Non-Technical Challenges

- *Time Management Between Phases*

  Balancing in-depth experimentation with Enigma and transitioning to scanner-based development was difficult. The research-heavy start consumed more time than expected, and I had to adjust my workflow mid-way to ensure completion of the second phase.

- *Limited Prior Experience with Security Tools*

  While I had a basic understanding of vulnerability scanners, working directly with raw outputs and handling their configurations introduced a steep learning curve. I had to self-study documentation and test configurations repeatedly to understand what each scanner contributed to the pipeline.

5 Suggestions and Conclusions

Better Community Documentation for Security Tools

Projects like Enigma and ZAP lack beginner-friendly guides for setup and use. Future trainees would benefit greatly from curated tutorials or internal documentation that outlines installation, typical errors, and sample configurations.

More Prompt Engineering Training

Working with GPT models is not just about connecting APIs—it requires structured thinking, task chaining, and understanding LLM limitations. Introducing trainees to prompt engineering techniques early in the project would improve output quality and reduce debugging time.

Extend Exposure to Collaborative Tools

The internship was solo in nature, and adding some collaborative code review sessions or joint debugging hours would create learning opportunities from others' workflows and tool usage.

Conclusion and Personal Growth

This internship was a unique opportunity to explore the intersection of cybersecurity and artificial intelligence in a real-world context. The experience taught me how to approach complex systems iteratively, how to research and deploy security tools, and how to integrate modern AI capabilities for enhanced usability and automation.

Skill Development: I improved significantly in Python scripting, Docker, scanner integration, and GPT prompt engineering. I also learned how to document and structure projects for clarity and reproducibility.

Problem-Solving Confidence: Encountering ambiguous errors and inconsistent outputs strengthened my debugging and critical thinking skills. I became comfortable resolving configuration-level issues and working with AI as a tool rather than a black box.

Time Management: Managing two separate phases—research and implementation—taught me to allocate time wisely, set project scopes realistically, and prioritise deliverables over perfection.

Overall, my time at Beno Technologies solidified my interest in AI-augmented cybersecurity and gave me a strong foundation to continue building tools that make complex tasks more accessible and efficient.

Appendix A – *Daily Journal*

# June 10, 2025

Started my internship focused on AI-driven penetration testing solutions. I surveyed recent advances in VAPT automation using large language models (LLMs). Two tools stood out—XBow, which showed strong performance but was proprietary, and Enigma, which was open-source and community-driven. I decided to focus my exploration on Enigma due to its accessibility and integration with OpenAI models.

# June 11, 2025

Dove deeper into Enigma's SWE agent, which orchestrates GPT-based exploitation logic. Spent the day reading their documentation and GitHub issues to understand how prompts are managed internally. Reviewed their CTF performance data and tried to interpret why solve rates were relatively low. Also noted how Enigma sets up a terminal sandbox for secure GPT interaction.

# June 12, 2025

Focused on studying individual CTF writeups from the Enigma project page. Analysed how the AI agent interpreted challenges like open ports, misconfigurations, and reverse shells. Noted frequent failures in enumeration logic and weak performance in privilege escalation chains. The system showed promise but seemed to lack guidance in multi-step attacks.

# June 13, 2025

Today I explored the actual Enigma codebase to understand module structure and logic chaining. The SWE agent's orchestration pipeline was well-abstracted, but hardcoded assumptions made generalisation tricky. I considered how I could adapt or control its prompt flow externally. Took detailed notes to prepare for potential local deployment later.

# June 23, 2025

Mapped out an architecture diagram of Enigma's agent system and how it interfaces with OpenAI's API. Noticed that it treats terminal output linearly, with no clear session memory unless manually prompted. This insight gave me ideas for improving context retention by wrapping prompts in more structured guidance. I also recorded system-level requirements for setup.

# June 24, 2025

Finished outlining a personal implementation plan and environment setup strategy. Planned to use

Docker for sandboxing and Python venv for GPT scripting. I also noted an issue on the GitHub I would later use for testing Enigma’s performance. This wrapped up my research phase and prepared me for hands-on implementation.

# June 25, 2025

Cloned the Enigma repository and began environment setup. Installed required dependencies in a virtual environment and attempted to build a Docker sandbox for command execution. Faced several Python-related conflicts and base image issues when setting up the shell execution container. Logged all errors for troubleshooting and started tweaking the Dockerfile.

# June 26, 2025

Resolved Docker socket and permission errors by adjusting container runtime flags and mapping host volumes correctly. Got the sandboxed terminal environment up and running inside Docker. Successfully connected OpenAI GPT-4o keys via .env and confirmed communication with the model. First trial runs initiated, but the output was inconsistent. Also studied a few basic shell tasks such as ls, cat, and file traversal to observe behaviour.

# June 30, 2025

Tested a basic directory traversal challenge in a local CTF box. Enigma could navigate directories, but got stuck in situations requiring creative chaining or payload crafting. I manually walked it through the next steps, confirming that human-augmented prompting significantly improves success. This highlighted the need for guided prompt templates.

# July 01, 2025

Created a set of reusable prompt templates for common tasks like enumeration, file reading, and privilege escalation. Each prompt provided additional context and task constraints to reduce GPT ambiguity. I saved these templates into a local YAML config for easier chaining later. This change made the model more consistent in shell execution.

# July 02, 2025

Ran a more advanced test involving misconfigured file permissions. The agent was able to list and read from sensitive files but failed to escalate privileges. I intervened manually and used GPT to explain why the escalation path failed—this reinforced the educational potential of combining AI with guided lab setups. Logged both success and failure cases for final reporting.

# July 03, 2025

Enhanced the sandbox execution flow to capture both stdout and stderr for better GPT interpretation. This led to more accurate command chaining, especially when the model needed to correct itself after an error. The improved feedback loop helped GPT navigate recovery paths when its assumptions failed. Began writing documentation on sandbox deployment.

# July 04, 2025

Packaged my sandbox setup into a reproducible deployment script using Bash and Python. This included Docker build commands, OpenAI key handling, and GPT environment initialisation. I finalised the setup environment and archived working configuration files. The Enigma pipeline was now stable and ready for controlled exploitation experiments.

# July 07, 2025

Today, I completed basic functionality testing of the Enigma setup using a controlled local environment. The system could handle simple enumeration tasks and responded correctly to straightforward prompts. While I didn't go into full exploitation chains or complex privilege escalation, the setup validated that the environment was usable for further testing or experimentation. I saved working configurations and logs and began shifting focus toward researching vulnerability scanners for the next phase of the project.

# July 08, 2025

I spent today transitioning from the Enigma-based exploitation work into vulnerability scanning research. This involved reviewing the goals for the second half of the project and identifying which FOSS tools would integrate well with AI for automated reporting. I explored several scanners, including Nuclei, Nikto, and OWASP ZAP, assessing their output formats and documentation. This day was mainly focused on understanding which tools were scriptable and would generate clean data that could be parsed and explained by an LLM.

# July 09, 2025

Completed the integration of both Nuclei and Nikto into the scanner pipeline. Outputs were directed to versioned files and parsed using custom scripts (parse_nuclei.py and parse_nikto.py). I initiated the development of an AI-based markdown report generator that would summarise findings using GPT-4o. Results were decent, but too verbose—so prompt tuning became the next priority.

# July 10, 2025

Today I worked on improving AI summarisation using the ai_explainer.py module. I tested the OpenAI ChatCompletion API with structured messages that included titles, severity levels, and vulnerability descriptions. This setup enabled the AI to generate clear and readable security explanations, although some redundancy was present. I began refining the prompt structure and added fallback logic for failed API calls.

# July 11, 2025

I linked everything together in main.py, a central script that automates scanning, parsing, AI explanation, and markdown report generation. The script now dynamically names output files using versioning logic and de-duplicates results across scanners. I ran successful scans against known vulnerable targets and received coherent reports with grouped findings and summaries. This was the first complete end-to-end test of the system.

# July 14, 2025

Refined report formatting by organising the AI output under each vulnerability title in markdown format. The output now includes affected URLs, scanner names, and severity indicators, followed by an AI-written explanation. To improve readability, I added headers and ensured line breaks were handled cleanly. This small enhancement made a big difference when reviewing long reports.

# July 15, 2025

Expanded the parser functionality to include support for ZAP scanner outputs. While ZAP integration was more complex due to its JSON structure, the parse_zap.py script successfully extracted alerts and classified them by severity. Although ZAP wasn't fully integrated in production runs, its support was documented and can be activated via a CLI flag in main.py. This modular approach keeps the system flexible.

# July 16, 2025

Tested the complete setup on several test targets and validated the accuracy of both scans and AI summaries. I benchmarked the system's performance and found that Nuclei was faster but sometimes too aggressive, while Nikto produced more generic findings. AI explanations helped bridge this gap by giving context and prioritisation to otherwise noisy output. I noted this observation in the documentation.

# July 17, 2025

I completed the last major round of testing for the AI-powered scanning and reporting framework. The pipeline—combining Nuclei, Nikto, and GPT—was confirmed to be working reliably on test targets. I spent the day validating input-output flow across modules and reviewing some example markdown reports. No major bugs appeared during runtime, and all scanners responded properly. With that, I considered the core functionality of the project completed.

# July 18, 2025

On the final day of my internship, I focused on ensuring that all project files were in place and usable from a development perspective. I ran the full system one last time to confirm stability and consistency. While I did not produce extended documentation or package it into a full release, the codebase was clean and modular, with all working scripts included in the folder which I added to a repository on GitHub. I noted a few possible future enhancements, like improving ZAP stability and simplifying prompt structures, but overall, the framework achieved the intended goal. This wrapped up my internship, which gave me hands-on exposure to LLMs in security tooling and practical scanner integration.
