FINAL YEAR PROJECT RESEARCH REPORT

Project Title: IGNIS, Vulnerability Management Personal Assistant

Created By

Khairul Amirin Bin Syahrean

Student Number: C00265680

4th Year (Hons) Cybercrime and IT Security

South East Technological University Carlow Campus

Supervised by

Richard Butler

April 19, 2024

Abstract

This is the final report of Ignis, a project for creating AI-powered personal assistant to enhance IT security operations by managing vulnerabilities. Utilizing technologies like LangChain, GPT-3.5 Turbo, and Streamlit, this initiative resulted in a chatbot capable of real-time interactions and tasks such as vulnerability assessment and automated communications.

This document showcases the deliverable, my experience in the project and problems encountered during development. Finally, I would present the key lessons obtained from this endeavour. These lessons include understanding vulnerability management, limits of AI, and data management.

Acknowledgements

I would like to thank Richard Butler, my project supervisor for overseeing my project development and testing my project prototypes. His invaluable guidance and unwavering support throughout the development of this project. His insights and expertise were crucial in steering the project towards success, and his encouragement kept me motivated during challenging periods.

I would also like to thank Joseph Kehoe, the 4th year project manager for Cybercrime and IT Security students. His advice and recommendations always ensure I keep my project in line with proper guidelines.

Additionally, I would like to thank my peers from 4th Cybercrime and IT Security for their support.

Contents

Introduction

In this final report, I document the culmination of my final year project, IGNIS, a vulnerability management application with AI capabilities. Throughout the duration of this project, I have gained invaluable insights and knowledge in various areas of software design, development, and application deployment.

The journey from conceptualization of an AI chatbot to the final deployment of the application was filled with unexpected challenges, learning opportunities, and significant milestones with each one giving me a confidence boost to reach the end. This experience has not only enhanced my technical skills in app building and utilizing LLMs but also fostered a deeper understanding in developing solutions for real-world IT security scenarios.

This report aims to not only showcase the final product and test runs conducted, but also to discuss the key learnings and takeaways that have emerged during the development of IGNIS. It serves as both a reflection on the achievements I obtained, the challenges encountered, and the key steps I took throughout the project lifecycle.

Project Description

Ignis is an AI-powered chatbot that acts as a personal assistant focused on overseeing and handling vulnerability management for a company's IT security infrastructure. This assistant aims to facilitate real-time, user-friendly interactions by providing immediate insights on threats and vulnerabilities, enhancing IT security operations within the organization.

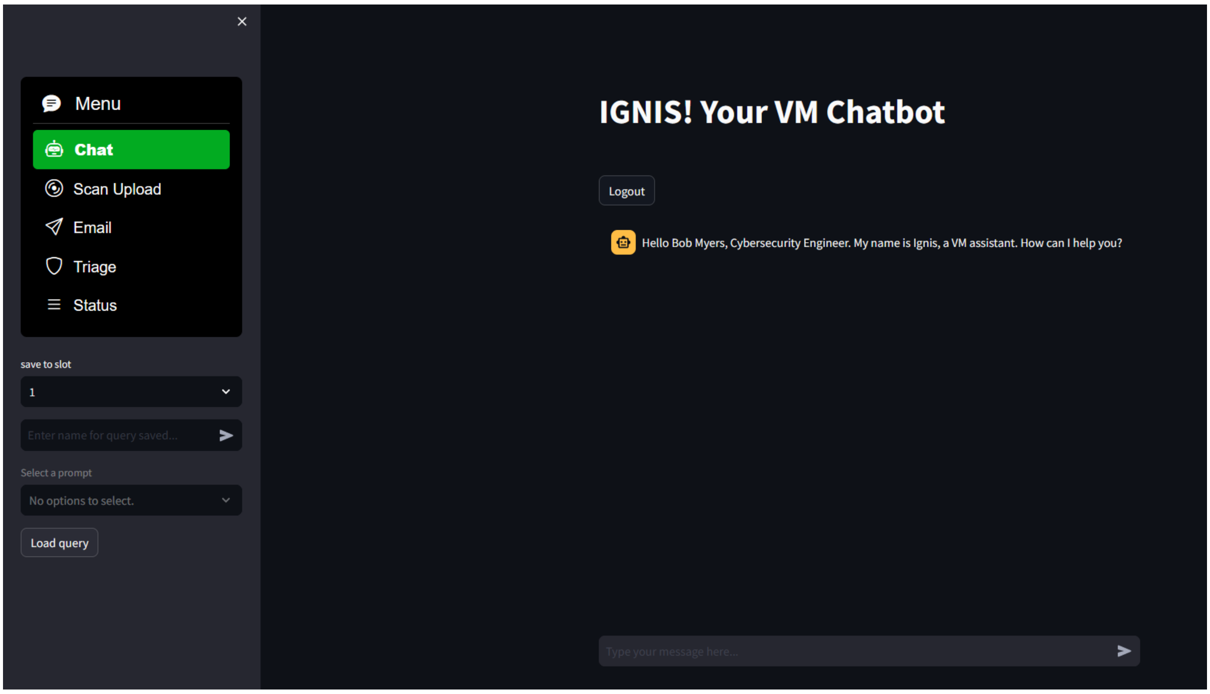
Notable Technologies Used:

- **LangChain:** Utilized as the fundamental AI technology to empower the chatbot functionalities within the application, leveraging advanced NLP capabilities to understand and respond to user queries regarding IT security vulnerabilities.
- **GPT 3.5 Turbo:** The core language model offers robust AI-powered responses, making the personal assistant capable of human-like interactions and providing precise vulnerability insights extracted from integrated security tool data.
- **Streamlit:** Primary backbone for developing the user interface. Streamlit's web framework simplifies the creation of user-friendly, interactive tools that allow cybersecurity personnel to seamlessly interact with the AI chatbot and retrieve vulnerability reports.
- **ChromaDB:** designed to effectively manage and retrieve large sets of vectors generated by the AI for purposes such as searching, comparing, and analysing data. The AI chatbot can quickly access historical data, optimizing the process of identifying and responding to security vulnerabilities and threats in an organization's IT infrastructure. This results in theory, a more informed and efficient vulnerability management system, significantly bolstering IT security protocols and response strategies.

To run the project:

1. Install python packages on requirements.txt.
2. Have XAMPP installed and import ignis.sql found zipped with project files.
3. Run menu.py on the console terminal of the IDE.

Project Deliverable



IGNIS! Your VM Chatbot

Hello Bob Myers, Cybersecurity Engineer. My name is Ignis, a VM assistant. How can I help you?

save to slot: 1

Select a prompt: No options to select.

Figure 1: Project Frontpage

The main UI takes majority of the screen to the right. There is a sidebar to the left that can be minimized for more space for the main console.

Use cases:

- Users can query for security status of the network environment. (Chat)
- Upload scan reports to the database (Scan Upload)
- Email remediation teams if needed. (Email)
- Assign vulnerabilities to remediation teams (triaging)
- List vulnerabilities, statuses and assigned remediation teams. (Status)

Front End Graphical Interface

MENU PAGE

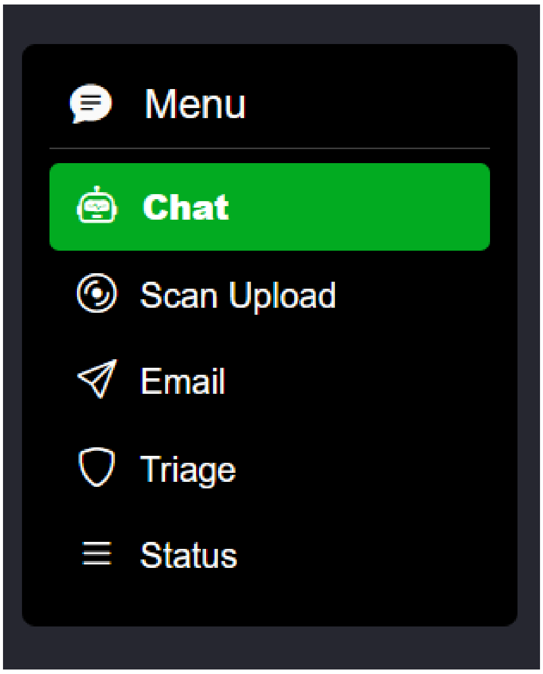


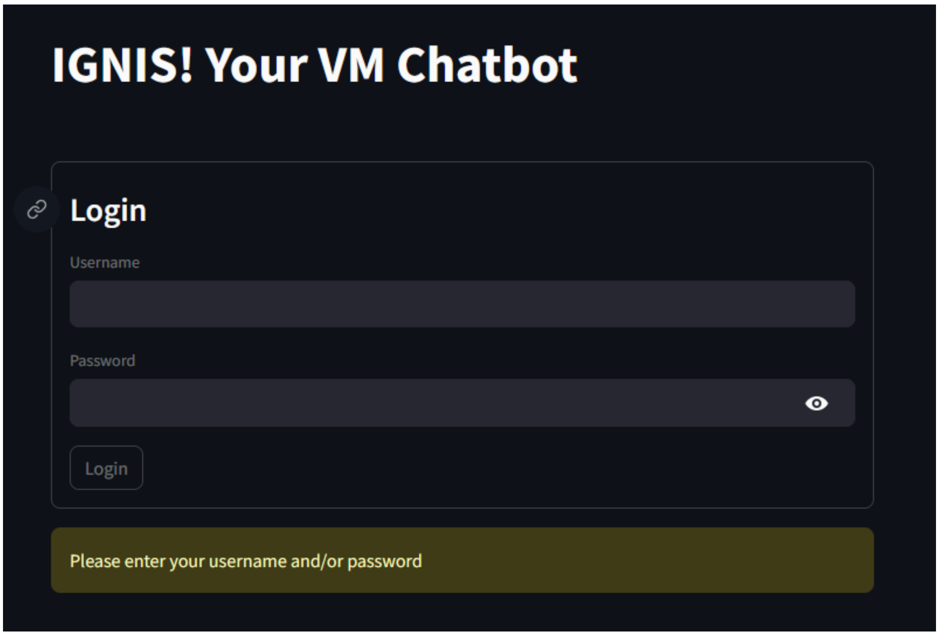
Figure 2: Sidebar Menu

Creating the Sidebar Menu:

- The script creates a sidebar menu using the option_menu function, part of streamlit’s many embedded functions.
- The menu includes options each associated with an icon and is styled with different CSS properties such as background color, text color, and font size.

Based on the selected menu option, the script calls the corresponding app function defined in the imported modules. (scan, chat, emailing, triage, list)

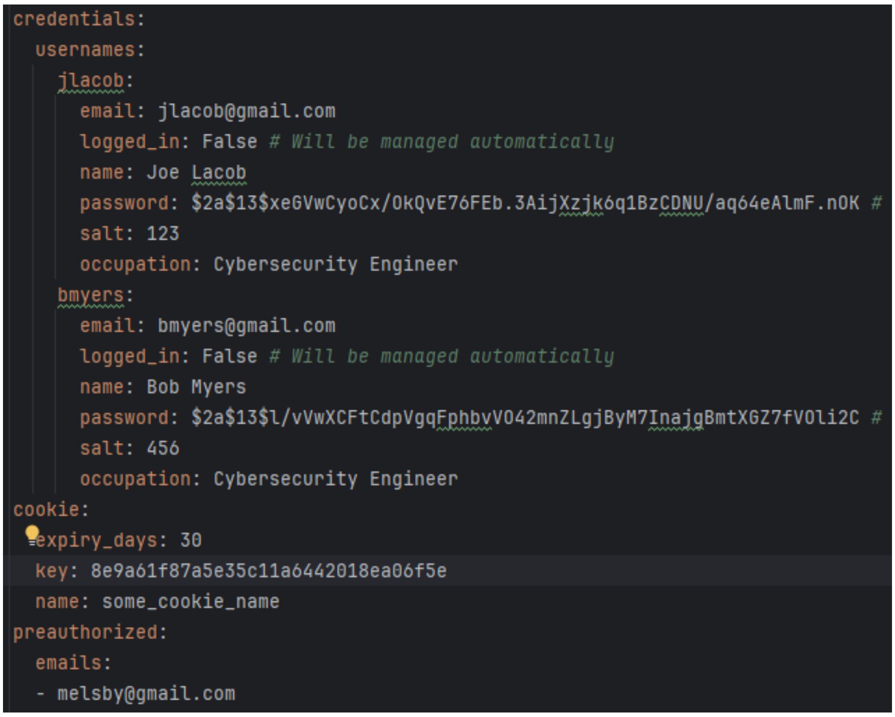
LOGIN PAGE



Please enter your username and/or password

Figure 3: Authentication Widget

I imported a package called streamlit_authenticator. Users must enter their username and password. It provides basic authentication service, denying access to Ignis if crendentials are wrong or is empty.



```yaml
credentials:
  usernames:
    jlacob:
      email: jlacob@gmail.com
      logged_in: False # Will be managed automatically
      name: Joe Lacob
      password: $2a$13$xeGVwCyoCx/OkQvE76FEb.3AijXzjk6q1BzCDNU/aq64eAlmF.nOK #
      salt: 123
      occupation: Cybersecurity Engineer
    bmyers:
      email: bmyers@gmail.com
      logged_in: False # Will be managed automatically
      name: Bob Myers
      password: $2a$13$l/vVwXCFtCdpVgqFphbvVO42mnZLgjByM7InajgBmtXGZ7fVOli2C #
      salt: 456
      occupation: Cybersecurity Engineer
cookie:
  expiry_days: 30
  key: 8e9a61f87a5e35c11a6442018ea06f5e
  name: some_cookie_name
preauthorized:
  emails:
  - melsby@gmail.com
```

Figure 4: All user information stored on a .yaml file.

The streamlit_authenticator package interacts with a yaml file to manage authentication. Passwords are hashed using bcrypt function. Storing passwords in plaintext puts a massive risk on application confidentiality. For future implementation I would set up a separate server specifically for authenticating users.

Note: Originally I had planned to add a registration page and change password feature but felt it was unnecessary as cybersecurity users should be registered with administrators handling the application's backend rather than self-registration.

CHAT PAGE

IGNIS! Your VM Chatbot

Hello John Smith, Cybersecurity Engineer. My name is Ignis, a VM assistant. How can I help you?

Hello Ignis

Hello John Smith! How can I assist you today with your cybersecurity needs?

Figure 5: Chat Interface

The main interface for cybersecurity users to interact with the OpenAI API.

The user can enter their query in the chat input field provided at the bottom of the page. The Ignis chatbot knows the user’s name and associated occupation, tailoring the user’s needs according to said information.

Responses take approximately 5 – 8 seconds, having to process the user’s query, whole conversation history and attached prompt, to be sent to the API.

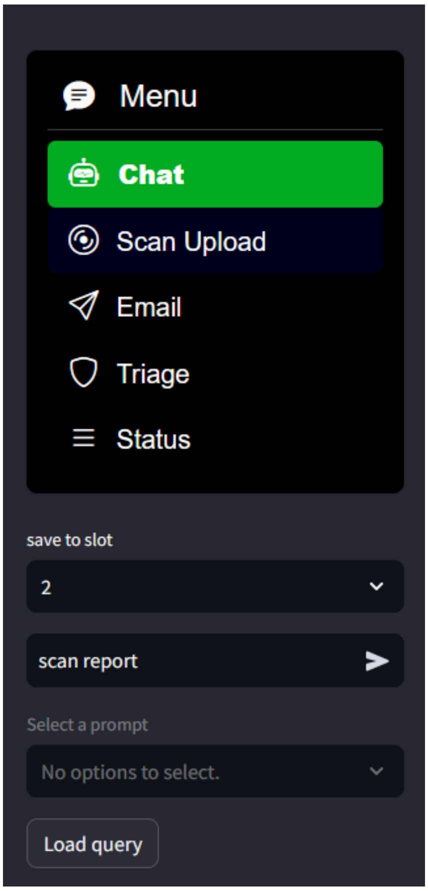


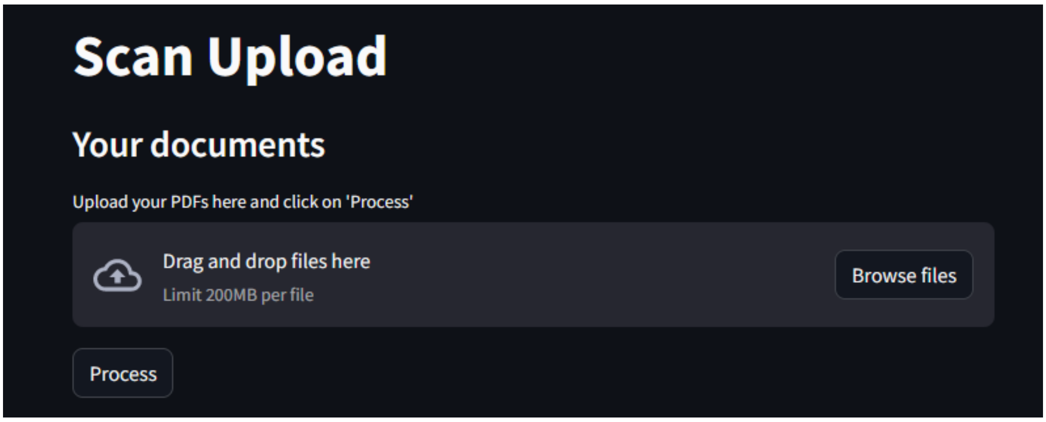
Figure 6: Save and Load Widgets

There is an option for users to save and load queries to make interaction with the bot a lot efficient. These options appear below the sidebar menu on the chat page.

If the user selects a prompt from the dropdown menu and clicks the "Load query" button, the code retrieves the corresponding query and prompt from the MySQL database and sets the user query to a predefined message containing the loaded query.

If the user enters a name for the query in the sidebar and clicks the arrow button, Ignis will save the last message sent by the user to the SQL database to be used in the future.

SCAN REPORT UPLOAD PAGE


Scan Upload

Your documents

Upload your PDFs here and click on 'Process'

Figure 7: Upload widget

As shown in the picture, a user can only upload a maximum size of 200MB per file. There is no hard limit on the database size itself. This means multiple files can be uploaded to the application.

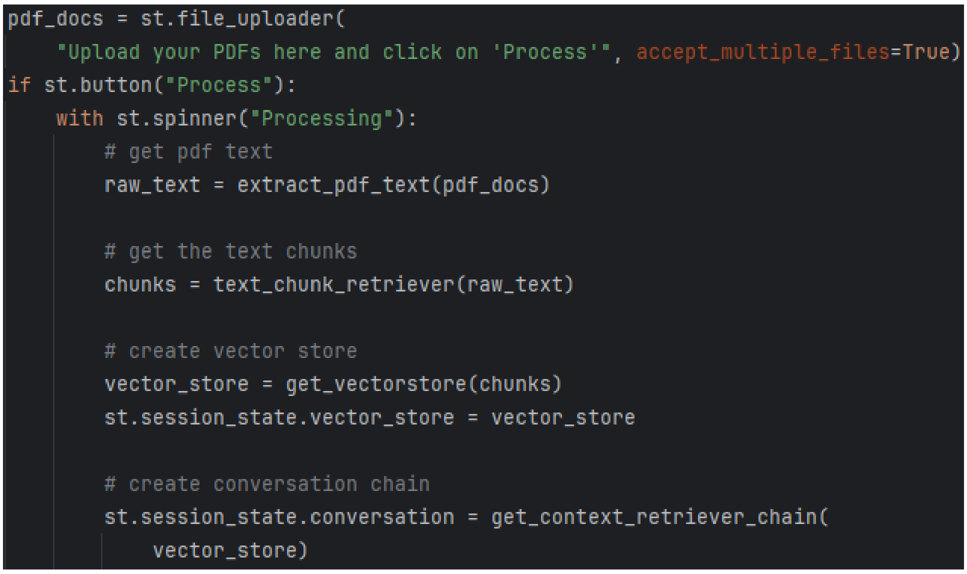
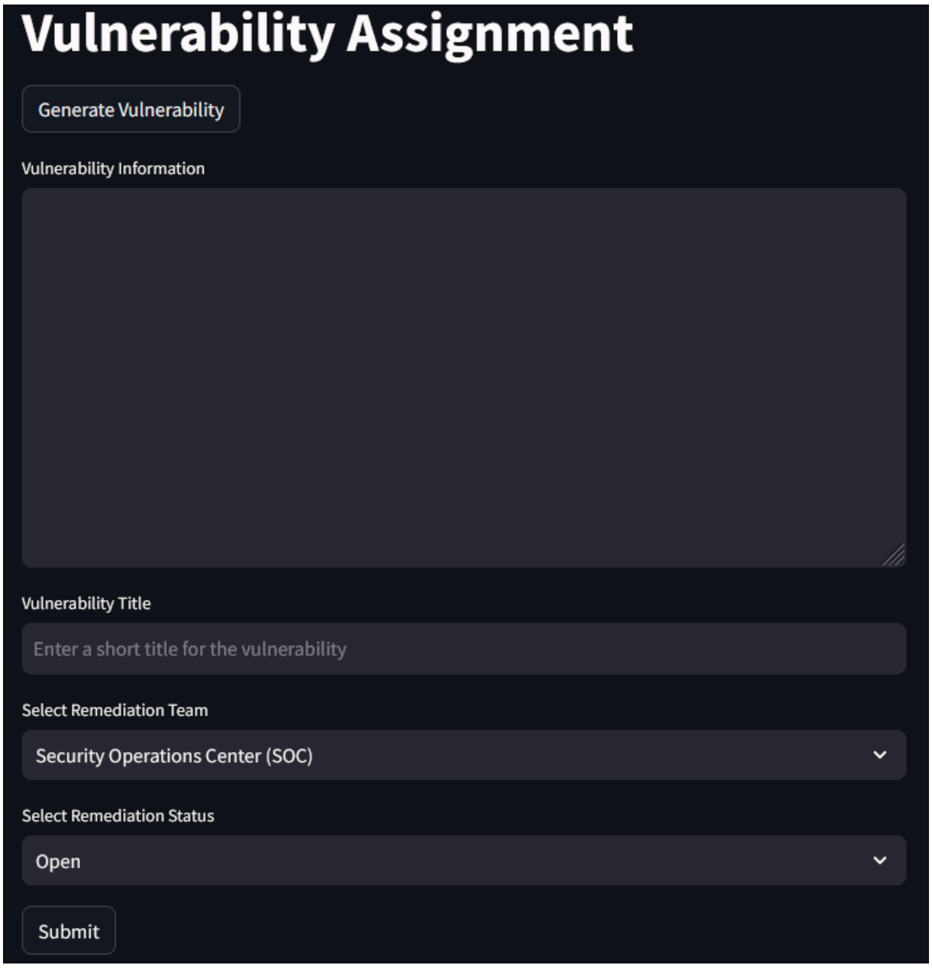

```python
pdf_docs = st.file_uploader(
    "Upload your PDFs here and click on 'Process'", accept_multiple_files=True)
if st.button("Process"):
    with st.spinner("Processing"):
        # get pdf text
        raw_text = extract_pdf_text(pdf_docs)

        # get the text chunks
        chunks = text_chunk_retriever(raw_text)

        # create vector store
        vector_store = get_vectorstore(chunks)
        st.session_state.vector_store = vector_store

        # create conversation chain
        st.session_state.conversation = get_context_retriever_chain(
            vector_store)
```

Figure 8: Code snippet of the upload function

Several functions are called from the main python file (chat.py). Text from the pdf file uploaded is extracted, segregated into chunks, and stored into ChromaDB database. These functions are broken down in backend section of the report.

VULNERABILITY ASSIGNMENT PAGE



Vulnerability Assignment

Vulnerability Information

Vulnerability Title

Select Remediation Team: Security Operations Center (SOC)

Select Remediation Status: Open

Figure 9: Vulnerability Assignment page

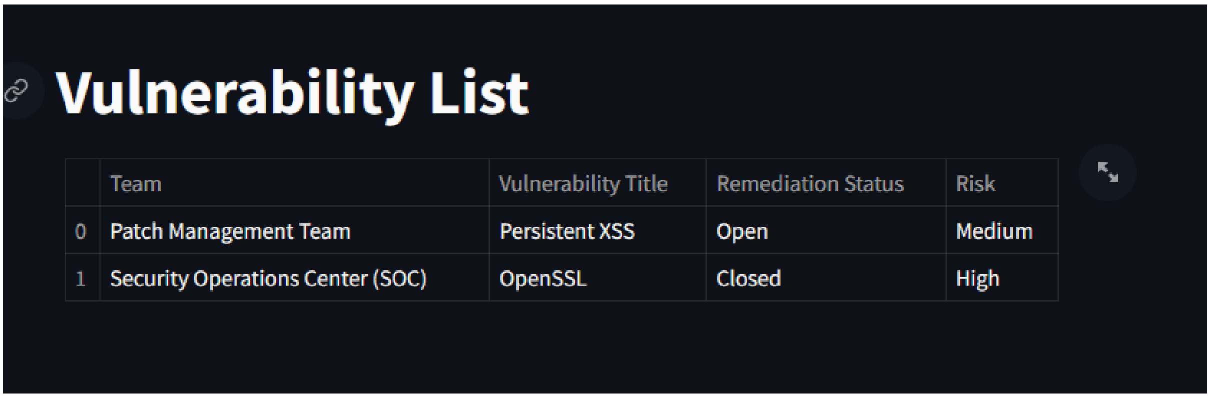
Generating Vulnerability:

When the "Generate Vulnerability" button is clicked, it uses an OpenAI model to identify the main vulnerability and list relevant information. The vulnerability information is displayed in a text area.

Upon clicking the "Submit" button:-

- Connects to a local MySQL database.
- Inserts the vulnerability title, information, remediation team, and status into the 'vulnerabilities' table.
- Commits the changes and closes the database connection.
- Displays a success message confirming that the vulnerability information has been saved and assigned to the remediation team.

VULNERABILITY STATUS PAGE



Vulnerability List

	Team	Vulnerability Title	Remediation Status	Risk
0	Patch Management Team	Persistent XSS	Open	Medium
1	Security Operations Center (SOC)	OpenSSL	Closed	High

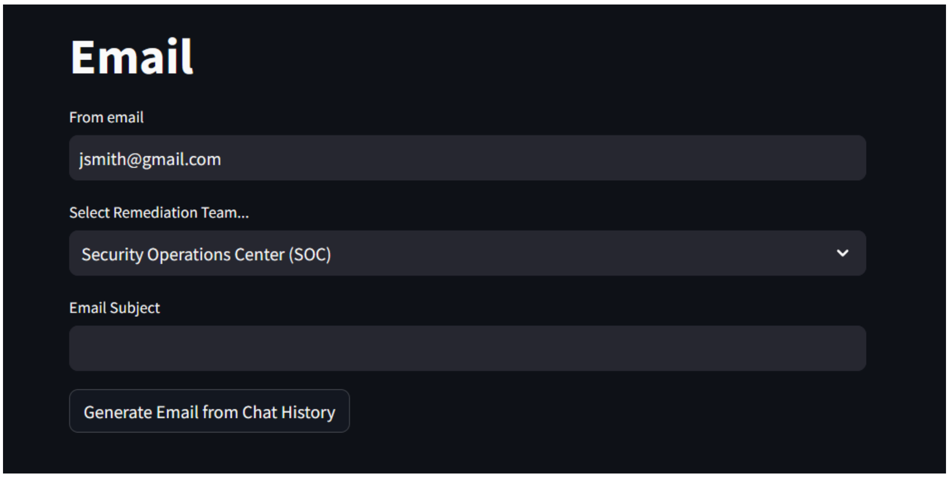
Figure 10: Table showing assigned vulnerabilities

Initializes a list to display all vulnerability information. If there is vulnerability data available, it displays it in a table with columns:

- Team
- Vulnerability Title
- Remediation Status
- Risk

Currently there is no option to delete entries from the list unless an administrator interacts with the database directly.

EMAIL PAGE



Email

From email

jsmith@gmail.com

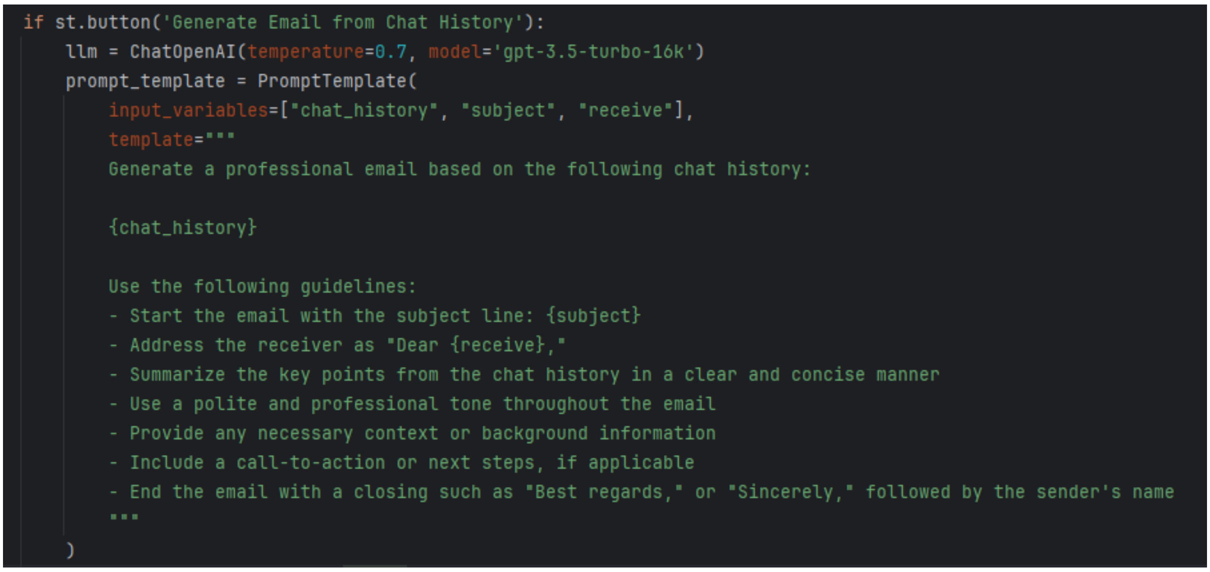
Select Remediation Team...

Security Operations Center (SOC)

Email Subject

Figure 11: Email widget

The initial setup involves displaying input fields for the sender's email, selecting a remediation team, and entering an email subject.



```
if st.button('Generate Email from Chat History'):
    llm = ChatOpenAI(temperature=0.7, model='gpt-3.5-turbo-16k')
    prompt_template = PromptTemplate(
        input_variables=["chat_history", "subject", "receive"],
        template="""
        Generate a professional email based on the following chat history:

        {chat_history}

        Use the following guidelines:
        - Start the email with the subject line: {subject}
        - Address the receiver as "Dear {receive},"
        - Summarize the key points from the chat history in a clear and concise manner
        - Use a polite and professional tone throughout the email
        - Provide any necessary context or background information
        - Include a call-to-action or next steps, if applicable
        - End the email with a closing such as "Best regards," or "Sincerely," followed by the sender's name
        """
    )
```

Figure 12: Email auto-generator

The green text is the prompt used to instruct GPT how the email should be formatted. When the "Generate Email from Chat History" button is clicked, the code initiates a prompt for generating the email content based on chat history to be sent to the OpenAI API.

The generated email response is then sent back and displayed on the email body text area for the user to review and possibly edit. A demo of this is shown on use case section.

Back End Architecture

GPT API

Halfway through the project's development I attempted to create fine-tuned model.

MODEL

ft:gpt-3.5-turbo-0125:kai-jones::92juz3o6 Succeeded

Job ID ftjob-b9QWgqHTwHX8HPxD0dBGoSPo

Base model gpt-3.5-turbo-0125

Output model ft:gpt-3.5-turbo-0125:kai-jones::92juz3o6

Created at Mar 14, 2024, 6:10 PM

Trained tokens 4,968

Epochs 8

Batch size 1

LR multiplier 2

Seed -

Files

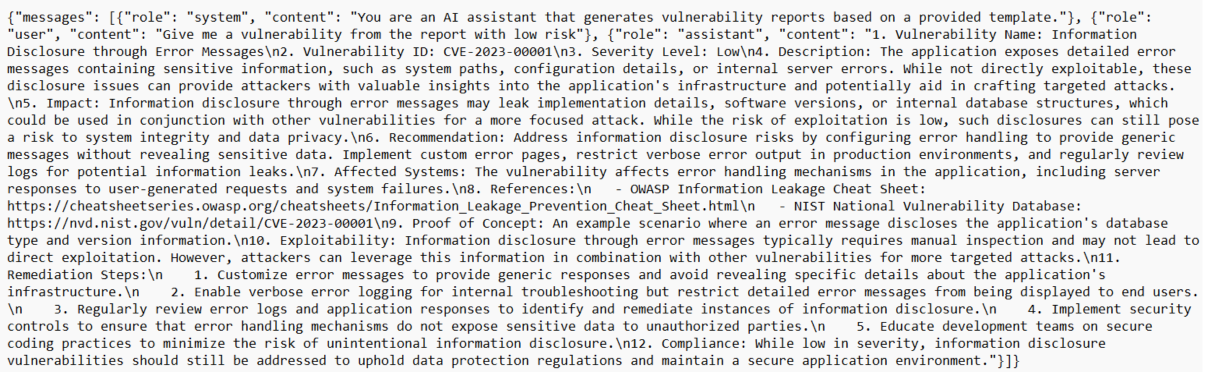
Training train.jsonl

Validation -

Training loss 0.1513

Figure 13: GPT model finished fine-tuning

I opt to train using OpenAI's dedicated fine-tune feature available on the website. There is also an option to conduct the training process using python code hosted on the local machine. OpenAI's website has a cleaner and easier to understand UI.



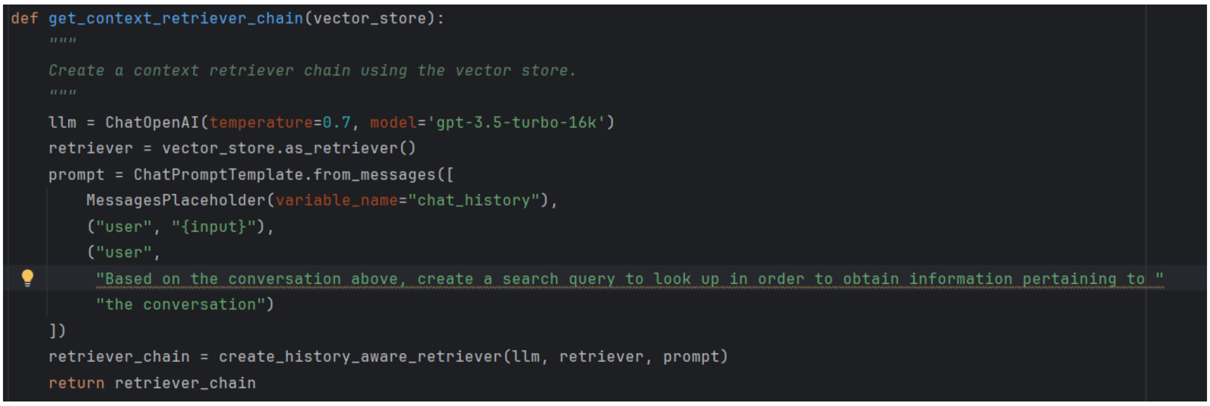
```
{"messages": [{"role": "system", "content": "You are an AI assistant that generates vulnerability reports based on a provided template."}, {"role": "user", "content": "Give me a vulnerability from the report with low risk"}, {"role": "assistant", "content": "1. Vulnerability Name: Information Disclosure through Error Messages\n2. Vulnerability ID: CVE-2023-00001\n3. Severity Level: Low\n4. Description: The application exposes detailed error messages containing sensitive information, such as system paths, configuration details, or internal server errors. While not directly exploitable, these disclosure issues can provide attackers with valuable insights into the application's infrastructure and potentially aid in crafting targeted attacks.\n5. Impact: Information disclosure through error messages may leak implementation details, software versions, or internal database structures, which could be used in conjunction with other vulnerabilities for a more focused attack. While the risk of exploitation is low, such disclosures can still pose a risk to system integrity and data privacy.\n6. Recommendation: Address information disclosure risks by configuring error handling to provide generic messages without revealing sensitive data. Implement custom error pages, restrict verbose error output in production environments, and regularly review logs for potential information leaks.\n7. Affected Systems: The vulnerability affects error handling mechanisms in the application, including server responses to user-generated requests and system failures.\n8. References:\n   - OWASP Information Leakage Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Information_Leakage_Prevention_Cheat_Sheet.html\n   - NIST National Vulnerability Database: https://nvd.nist.gov/vuln/detail/CVE-2023-00001\n9. Proof of Concept: An example scenario where an error message discloses the application's database type and version information.\n10. Exploitability: Information disclosure through error messages typically requires manual inspection and may not lead to direct exploitation. However, attackers can leverage this information in combination with other vulnerabilities for more targeted attacks.\n11. Remediation Steps:\n    1. Customize error messages to provide generic responses and avoid revealing specific details about the application's infrastructure.\n    2. Enable verbose error logging for internal troubleshooting but restrict detailed error messages from being displayed to end users.\n    3. Regularly review error logs and application responses to identify and remediate instances of information disclosure.\n    4. Implement security controls to ensure that error handling mechanisms do not expose sensitive data to unauthorized parties.\n    5. Educate development teams on secure coding practices to minimize the risk of unintentional information disclosure.\n12. Compliance: While low in severity, information disclosure vulnerabilities should still be addressed to uphold data protection regulations and maintain a secure application environment."}]}
```

Figure 14: Single sample of the training data used for fine-tuning the model.

Loading GPT Model:

The default model name and API key is stored in .env file. The GPT model is loaded using the ChatOpenAI class, specifying parameters such as temperature.

Ignis dynamically interacts with the GPT API based on the user's input and the context of the conversation, allowing the AI assistant to provide relevant and tailored responses in real-time.



```python
def get_context_retriever_chain(vector_store):
    """
    Create a context retriever chain using the vector store.
    """
    llm = ChatOpenAI(temperature=0.7, model='gpt-3.5-turbo-16k')
    retriever = vector_store.as_retriever()
    prompt = ChatPromptTemplate.from_messages([
        MessagesPlaceholder(variable_name="chat_history"),
        ("user", "{input}"),
        ("user",
         "Based on the conversation above, create a search query to look up in order to obtain information pertaining to "
         "the conversation")
    ])
    retriever_chain = create_history_aware_retriever(llm, retriever, prompt)
    return retriever_chain
```

Figure 15: Code snippet of context retriever

Retrieval and Generation Chains:

- The script creates a chain of retrievers and generators to interact with the GPT model.
- The *create_history_aware_retriever* function is used to create a retriever chain that considers the conversation history with the user and prompt when retrieving information from the ChromaDB database.
- Subsequently, a conversational retrieval-augmented generation (RAG) chain is created using the *create_retrieval_chain* function to combine the retriever chain with the generation logic.

1. **Handling User Input**:

- When a user inputs a message through the chat interface, the script calls the invoke method on the conversation RAG chain to generate a response based on the conversation context and the user's query.
- The script defines a context prompt template to provide context to the GPT model for generating responses.

- The response generated by the GPT model is then displayed back to the user in the chat interface.

History Management

```
# Conversation
for message in st.session_state.chat_history:
    if isinstance(message, AIMessage):
        with st.chat_message("AI"):
            st.write(message.content)
    elif isinstance(message, HumanMessage):
        with st.chat_message("Human"):
            st.write(message.content)
```

Figure 16: Code snippet for printing chat history

1. **Initialization**:

- When the user first interacts with the application, the initial messages are predefined.

2. **Storing Chat History**:

- The chat history is stored in the *st.session_state.chat_history* variable. This variable is initialized to hold messages exchanged between the user and the AI assistant.
- Messages are stored as instances of *AIMessage* and *HumanMessage* classes, distinguishing between messages generated by the AI and the user.

3. **User Input**:

- Users can input messages through a chat input box. The entered text is stored as *user_query* for processing.

4. **Processing User Input**:

- When a user submits a message, the script processes the input using functions that involve querying a database and generating responses based on the context.

5. **Displaying Chat Messages**:

- The chat messages are displayed in the conversation area of the chatbot interface.
- Messages exchanged between the AI and the user are shown with the sender's name (AI or Human) and the content of the message.

6. **Saving Conversations**:

- The script includes functionality to save the last three conversations to the MySQL database. This process involves updating existing records with new messages or inserting new records if no previous data is found.

7. **Loading Previous Queries**:

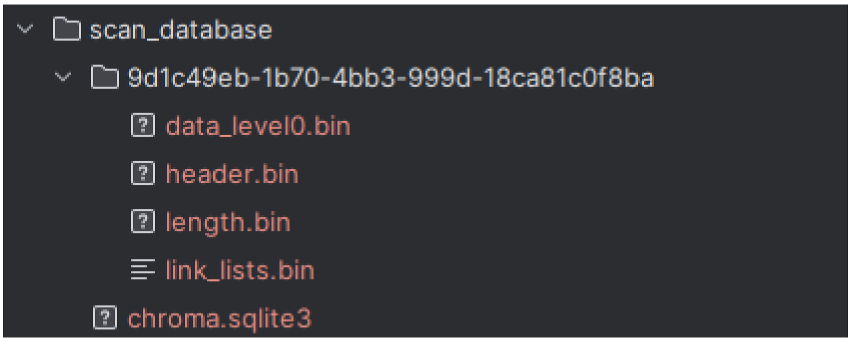
- Users have the option to load previously saved queries from the database. The script retrieves saved queries based on the user's selection and pre-populates the chat interface with the selected query for further interaction.

8. **Updating Chat History**:

- At each step of the conversation, chat history is updated with new messages added by the user and responses generated by the AI assistant.

Notes: Each time the API is called, the whole chat history, additional user queries and data pulled from vector database is sent to GPT 3.5 to be processed. The longer the overall text load and token count sent from base application, the more expensive the overhead and time needed to process an output. GPT does not have a built-in memory feature for custom LLM application building.

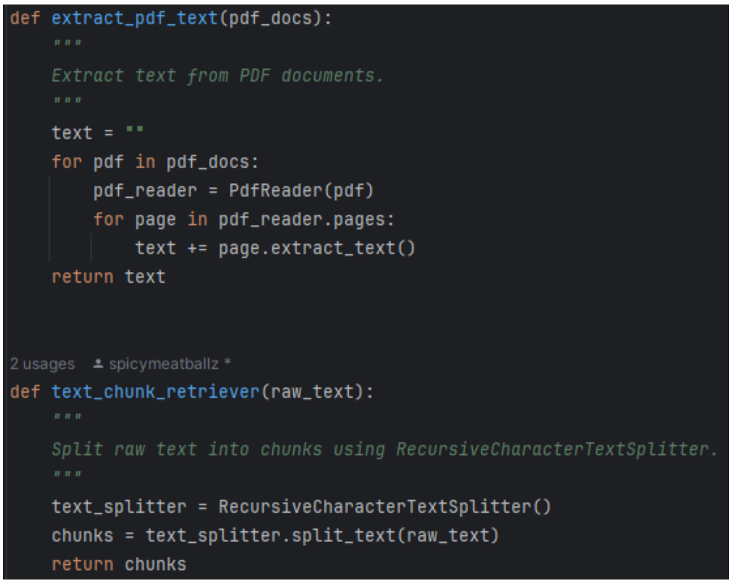
Vector Database



```
scan_database
  9d1c49eb-1b70-4bb3-999d-18ca81c0f8ba
    data_level0.bin
    header.bin
    length.bin
    link_lists.bin
  chroma.sqlite3
```

Figure 17: File of Hierarchy for ChromaDB database

ChromaDB is used as a vector database to store and retrieve embeddings for text chunks. Here is how Chroma is utilized as a vector database in the context of the given script:

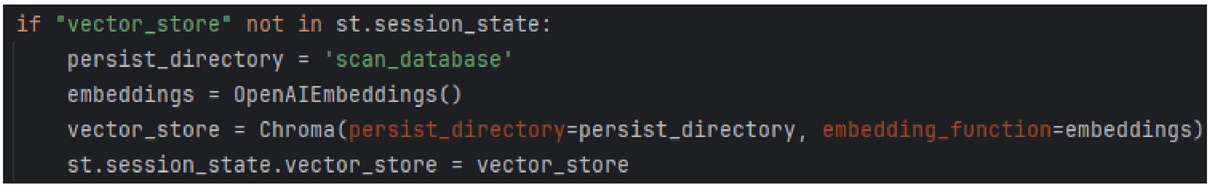


```
def extract_pdf_text(pdf_docs):
    """
    Extract text from PDF documents.
    """
    text = ""
    for pdf in pdf_docs:
        pdf_reader = PdfReader(pdf)
        for page in pdf_reader.pages:
            text += page.extract_text()
    return text


2 usages  spicymeatballz *
def text_chunk_retriever(raw_text):
    """
    Split raw text into chunks using RecursiveCharacterTextSplitter.
    """
    text_splitter = RecursiveCharacterTextSplitter()
    chunks = text_splitter.split_text(raw_text)
    return chunks
```

Figure 18: Text extractor and splitter

The first function (*extract_pdf_text*) goes through an uploaded file from the upload page and extracts all as a singular variable.

This variable called ‘raw text’ is then put through the second function. Text chunks are generated by splitting raw text into chunks using the *RecursiveCharacterTextSplitter* class.



```
if "vector_store" not in st.session_state:
    persist_directory = 'scan_database'
    embeddings = OpenAIEmbeddings()
    vector_store = Chroma(persist_directory=persist_directory, embedding_function=embeddings)
    st.session_state.vector_store = vector_store
```

Figure 19: ChromaDB creation function

1. **Creating and Persisting the Vector Store**:

- If not present, Ignis automatically creates a vector database called ‘scan_database’.
- The vector store created with Chroma is persisted to a directory using the persist() method. This allows the vector embeddings to be saved for future use without needing to recreate them each time the script runs.

```
def get_vectorstore(chunks):
    """
    Create or load a vector store using Chroma.
    """
    embeddings = OpenAIEmbeddings()
    persist_directory = 'scan_database'
    vector_store = Chroma.from_texts(texts=chunks, embedding=embeddings, persist_directory=persist_directory)
    vector_store.persist()
    vector_store = None
    vector_store = Chroma(persist_directory=persist_directory, embedding_function=embeddings)
    return vector_store
```

Figure 20: Code snippet for storing information

2. **Retrieving Vectors**:

- Ignis loads the vector store from the persistent directory using the Chroma constructor, specifying the directory path and the embedding function used.

3. **Utilizing Vectors for Context Retrieval**:

- The Chroma vector database stores all vital security scanning information of the organization. Data pulled is used to create a context retriever chain that considers the vector embeddings of text chunks when retrieving information relevant to the current conversation context.

4. **Dynamic Vector Handling**:

- The script dynamically manages the vector store created with Chroma to efficiently handle and retrieve vector embeddings for text chunks as needed during the conversation interactions.

SQL database

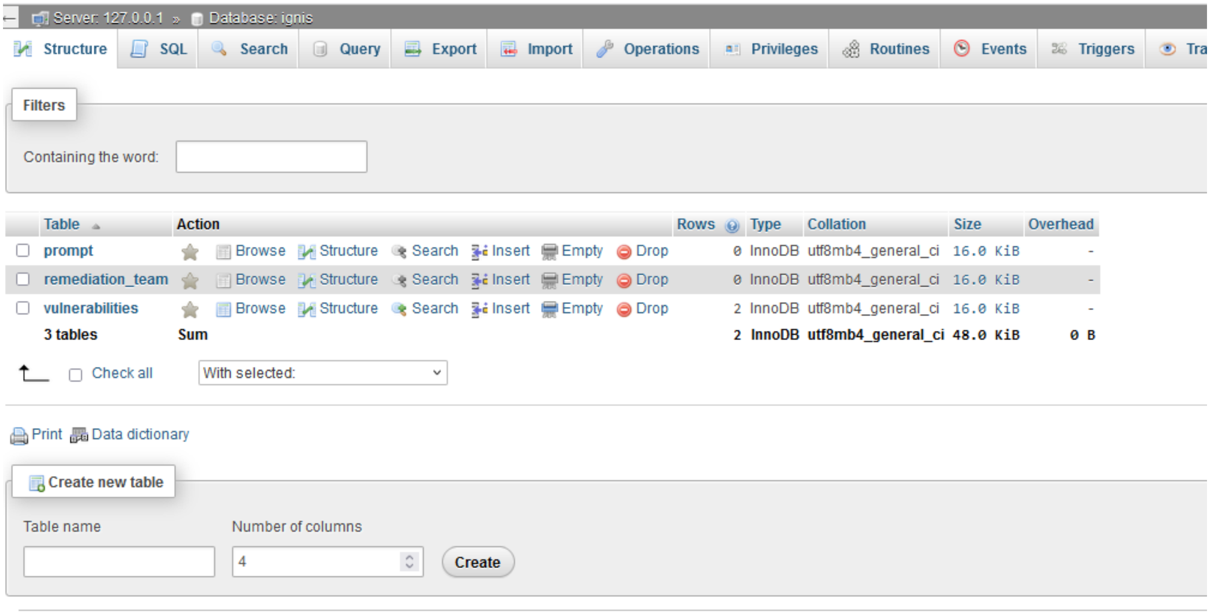


	Table	Action	Rows	Type	Collation	Size	Overhead
☐	prompt	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	16.0 KiB	-
☐	remediation_team	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	16.0 KiB	-
☐	vulnerabilities	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_general_ci	16.0 KiB	-
	3 tables	Sum	2	InnoDB	utf8mb4_general_ci	48.0 KiB	0 B

Table name

Number of columns: 4

Figure 21: MariaDB database

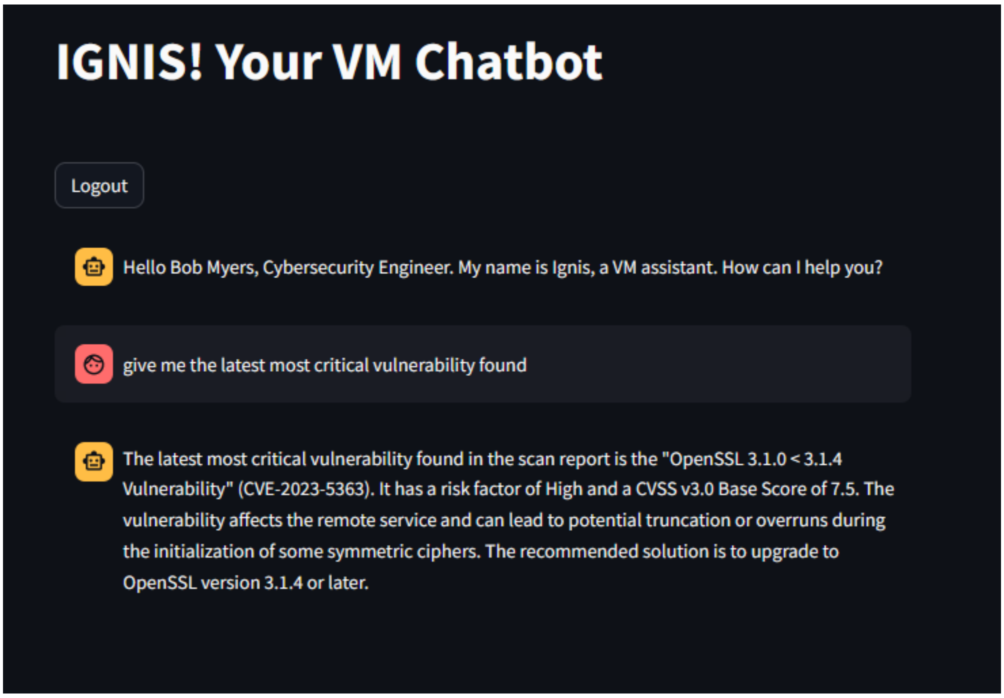
Two tables are mainly used for Ignis's operations.

- Prompt
  - Stores user's queries. These queries can be called to quicken bot interaction
- Vulnerabilities
  - Contains vulnerabilities and teams assigned to them.

The third table (remediation_team) is just for experimenting with setting up the SQL database to store credentials and is not used for the final project.

Application Testing and Use Case Examples.

1. Testing the API calling by giving user query.



IGNIS! Your VM Chatbot

Hello Bob Myers, Cybersecurity Engineer. My name is Ignis, a VM assistant. How can I help you?

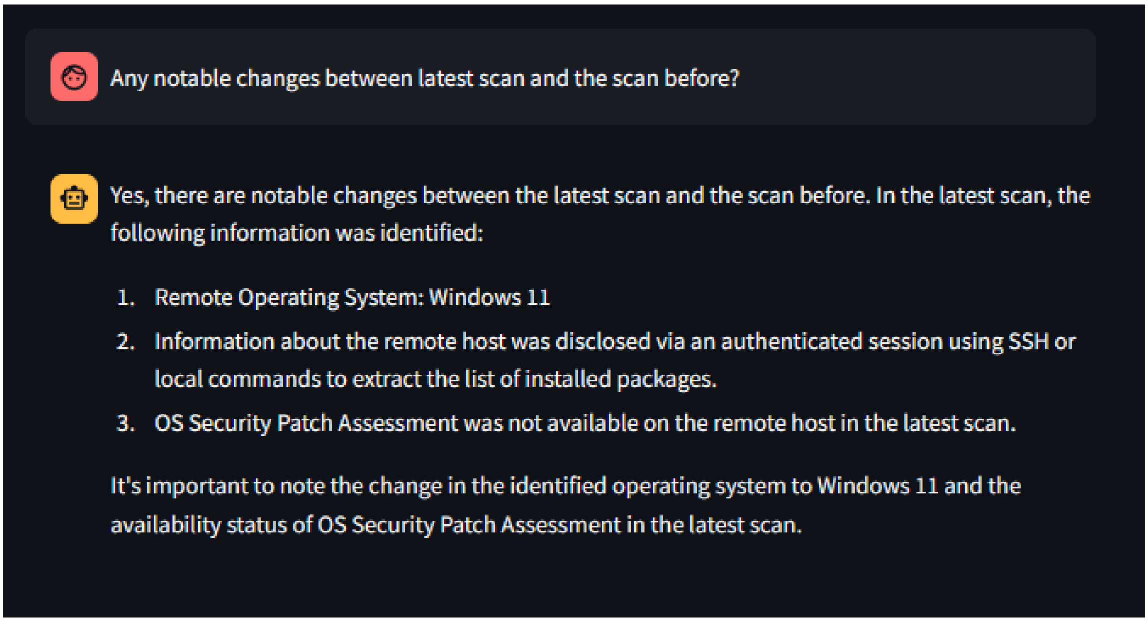
give me the latest most critical vulnerability found

The latest most critical vulnerability found in the scan report is the "OpenSSL 3.1.0 < 3.1.4 Vulnerability" (CVE-2023-5363). It has a risk factor of High and a CVSS v3.0 Base Score of 7.5. The vulnerability affects the remote service and can lead to potential truncation or overruns during the initialization of some symmetric ciphers. The recommended solution is to upgrade to OpenSSL version 3.1.4 or later.

Figure 22: Chatting

ChromaDB currently has a Nessus scan report conducted on my local machine. Ignis is using GPT to access the vector database, pull information based on whether the content matches the context of the user's query.

GPT would then process the information and turn it into a response to be outputted on the chatbot interface.



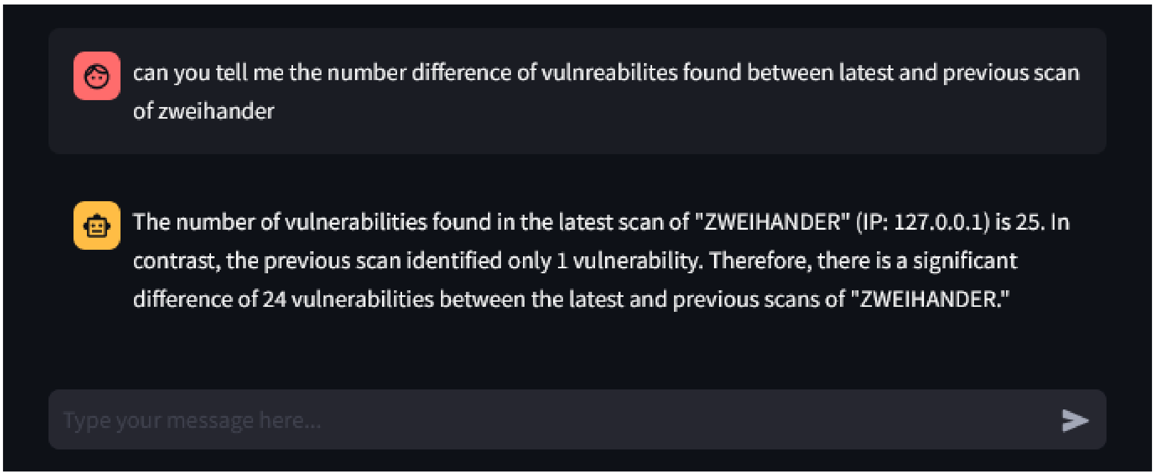
Any notable changes between latest scan and the scan before?

Yes, there are notable changes between the latest scan and the scan before. In the latest scan, the following information was identified:

1. Remote Operating System: Windows 11
2. Information about the remote host was disclosed via an authenticated session using SSH or local commands to extract the list of installed packages.
3. OS Security Patch Assessment was not available on the remote host in the latest scan.

It's important to note the change in the identified operating system to Windows 11 and the availability status of OS Security Patch Assessment in the latest scan.

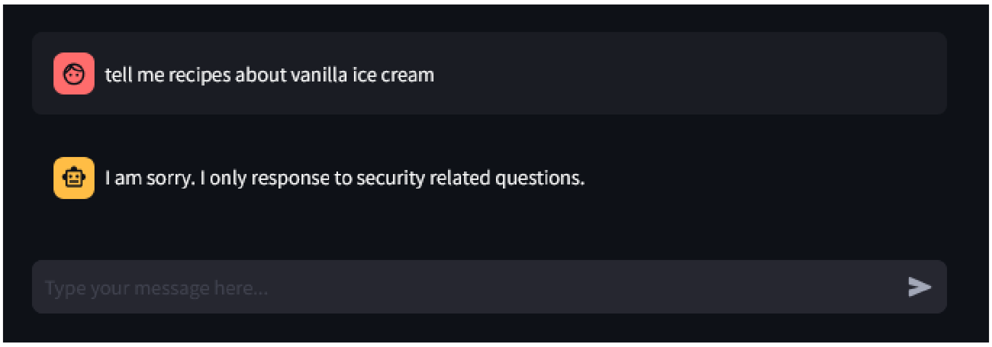
Figure 23: 2nd query attempted.



can you tell me the number difference of vulnreabilites found between latest and previous scan of zweihander

The number of vulnerabilities found in the latest scan of "ZWEIHANDER" (IP: 127.0.0.1) is 25. In contrast, the previous scan identified only 1 vulnerability. Therefore, there is a significant difference of 24 vulnerabilities between the latest and previous scans of "ZWEIHANDER."

Figure 24: 3rd query.

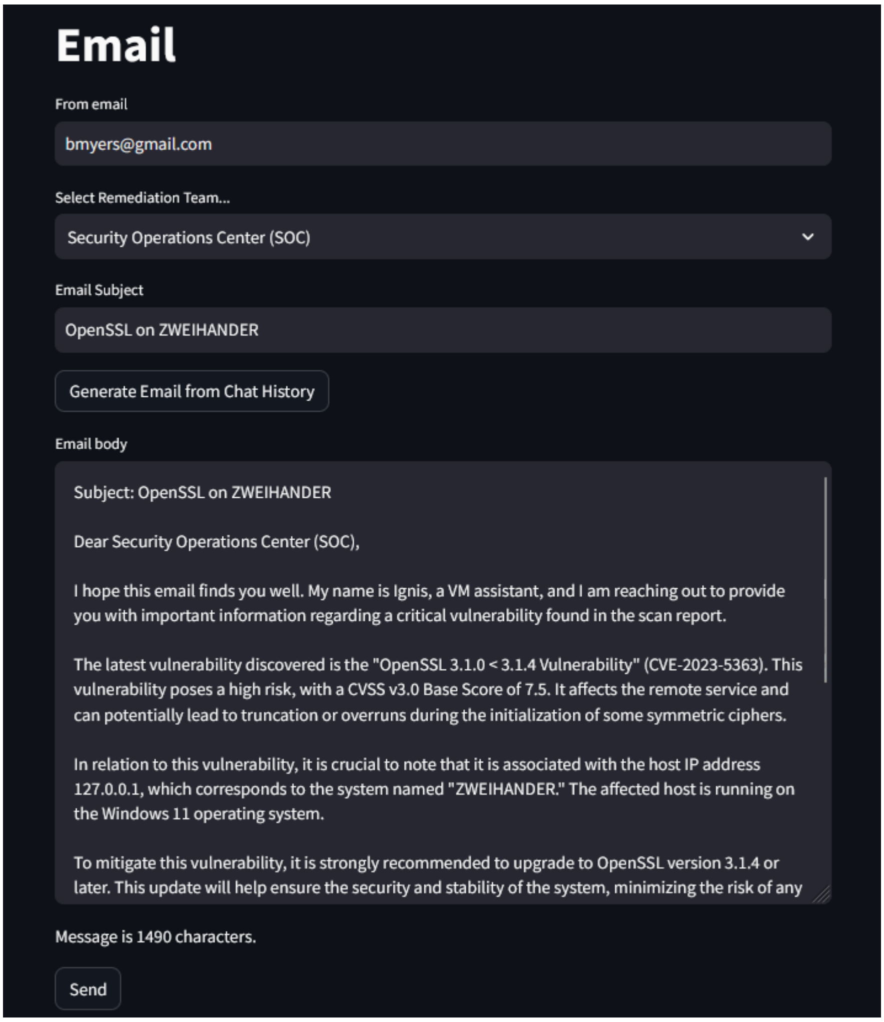


tell me recipes about vanilla ice cream

I am sorry. I only response to security related questions.

Figure 25: Random questioning.

Through fine-tuning and chat prompt, Ignis is less likely to answer non-security related questions.



From email: bmyers@gmail.com

Select Remediation Team...: Security Operations Center (SOC)

Email Subject: OpenSSL on ZWEIHANDER

Email body

Subject: OpenSSL on ZWEIHANDER

Dear Security Operations Center (SOC),

I hope this email finds you well. My name is Ignis, a VM assistant, and I am reaching out to provide you with important information regarding a critical vulnerability found in the scan report.

The latest vulnerability discovered is the "OpenSSL 3.1.0 < 3.1.4 Vulnerability" (CVE-2023-5363). This vulnerability poses a high risk, with a CVSS v3.0 Base Score of 7.5. It affects the remote service and can potentially lead to truncation or overruns during the initialization of some symmetric ciphers.

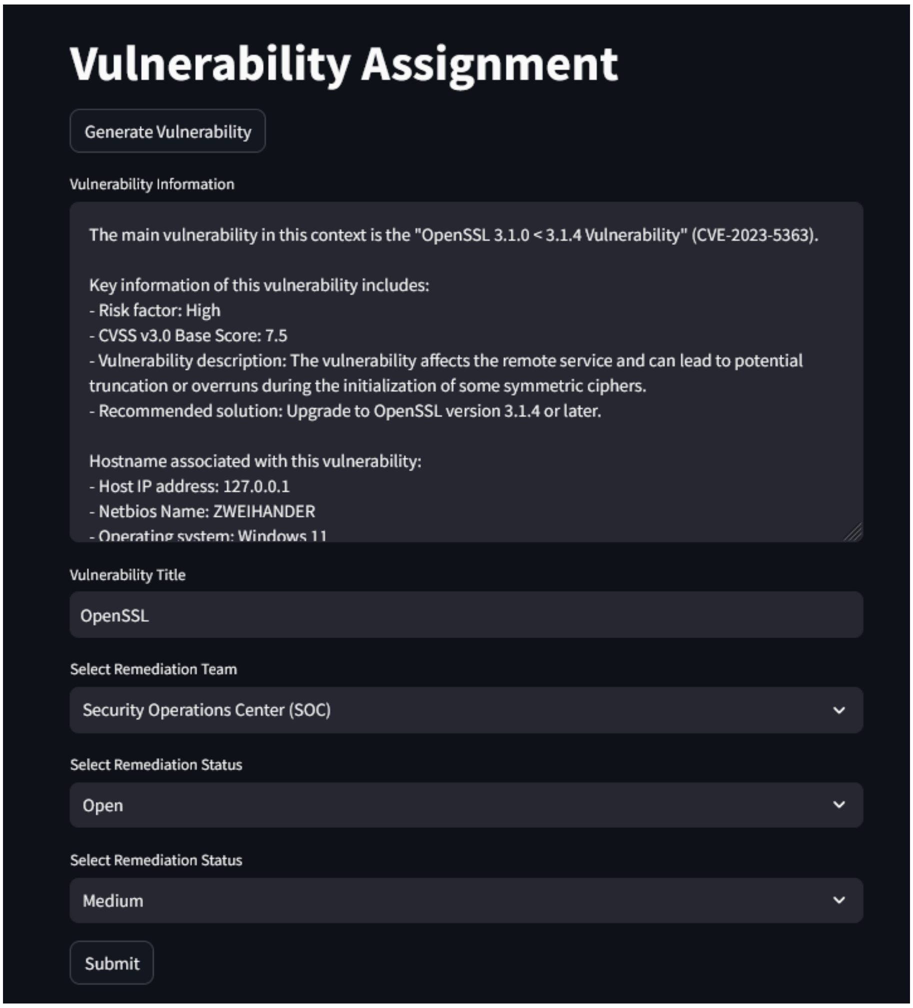
In relation to this vulnerability, it is crucial to note that it is associated with the host IP address 127.0.0.1, which corresponds to the system named "ZWEIHANDER." The affected host is running on the Windows 11 operating system.

To mitigate this vulnerability, it is strongly recommended to upgrade to OpenSSL version 3.1.4 or later. This update will help ensure the security and stability of the system, minimizing the risk of any

Message is 1490 characters.

Figure 26: Email widget.

The email is taken from the saved credentials of the user. User can select which remediation team to broadcast the email to.



Vulnerability Information

The main vulnerability in this context is the "OpenSSL 3.1.0 < 3.1.4 Vulnerability" (CVE-2023-5363).

Key information of this vulnerability includes:
- Risk factor: High
- CVSS v3.0 Base Score: 7.5
- Vulnerability description: The vulnerability affects the remote service and can lead to potential truncation or overruns during the initialization of some symmetric ciphers.
- Recommended solution: Upgrade to OpenSSL version 3.1.4 or later.

Hostname associated with this vulnerability:
- Host IP address: 127.0.0.1
- Netbios Name: ZWEIHANDER
- Operating system: Windows 11

Vulnerability Title: OpenSSL

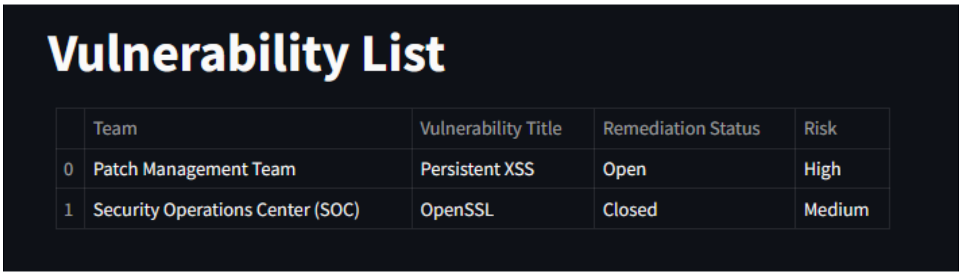
Select Remediation Team: Security Operations Center (SOC)

Select Remediation Status: Open

Select Remediation Status: Medium

Figure 27: Vulnerability Triaging

User can generate vulnerability description by clicking the button. Ignis will scan the conversation history, extract the main vulnerability, and turn it into a response to be saved into the database.



	Team	Vulnerability Title	Remediation Status	Risk
0	Patch Management Team	Persistent XSS	Open	High
1	Security Operations Center (SOC)	OpenSSL	Closed	Medium

Figure 28: Vulnerability List

All information is displayed in a table. This information can be used for other digital management platforms.

General Issues

Problems Encountered During Development:

Understanding Chat History and Tool Integration:

One of the significant challenges faced during the development process was effectively incorporating historical chat data into the application. The complexities surrounding the structure and real-time retrieval of data in live applications posed difficulties in seamlessly integrating chat history. Additionally, grasping the concept of chains within the LangChain framework required extra effort and understanding.

Adapting to the Open-Source Environment:

LangChain, being an open-source platform, undergoes frequent updates to its packages. This dynamic nature of the environment presented a challenge as deprecated packages needed to be replaced, and new ones had to be integrated continuously throughout the development process. Staying up to date with the latest changes and ensuring compatibility required constant attention and effort.

My biggest failure was knowing the limits of AI and its reasoning capability at the start of the project. With the assumption "AI is capable of running code by itself", I also overestimated LangChain's framework for AI apps as it is difficult to use it for commercial purposes.

Streamlit's Limitations in Customization:

While Streamlit provided a convenient framework for building the application, it posed limitations in terms of customization options. Creating a production-level application with desired customizations and advanced features proved to be challenging within the constraints of Streamlit's capabilities and lack of proper documentation for using the library.

Inconsistent Responses from GPT API:

During the initial stages of development, the responses generated by the GPT API were inconsistent and sometimes lacked coherence. To address this issue, efforts were made to refine the prompts and fine-tune the API, resulting in slightly improved response quality and relevance.

Inefficient Data Parsing Techniques:

Initially, the CSV format was used for data parsing, which led to suboptimal performance and efficiency. To overcome this limitation, the approach was switched to using PDF format. This change enabled more efficient parsing and seamless embedding of data into the vector database, enhancing overall system performance.

I initially used token and character-based text splitters to segregate chunks of text to be stored into the database. This created a set of data that is incoherent and difficult to query. I switched to recursive based text splitting as it uses certain guidelines to store a segment of text into the database and can be called easily by the OpenAI API.

AI Model Optimization

Was unable to properly fine-tune the AI model due to high costs and resource requirements. Some attempts are madeProper training of AI models demands significant capital investment and computational resources. Creating an ideal model requires numerous iterations, which proved to be a resource-intensive process.

Achieved/Not Achieved Goals

GUI

Expectation:

Release quality product including attractive GUI and features optimized for speed and ease of use.

Obtained:

The GUI is simple and easy to understand for new users. Users can switch pages using the menu on the sidebar of the webpage. The application connects to ChromaDB (Vector) and MariaDB (SQL) databases, which may affect the speed. The overall expectation for each request to the API and response is 6 to 8 seconds.

Advanced Chatbot Comprehension

Expectation:

Human-like interaction and responses.

Obtained:

With GPT-3.5, responses are typical to a GPT-style interaction. Usable but not entirely seamless and tends to add unnecessary information. The temperature is set to 0.7, slightly encouraging the API to provide more creative responses without affecting the meaning of the output. Using chat history as context and giving it to the chatbot helps to reduce hallucinations and maintain the direction of the conversation.

Comprehensive Asset Rundown

Expected:

Provide insight into assets using historical data (common vulnerabilities).

Obtained:

Security scan results are stored in a vector database. Ignis can differentiate between hosts and their vulnerabilities based on the date of scans. Its capability to form conclusions and inferences however between these scans is not reliable as GPT tends to avoid making conclusive statements. (Using GPT 4 alleviates this problem slightly but still inconsistent)

Scanning Capability

Expected:

Can execute Python scripts for running Nmap and Nessus functionalities by installing the appropriate libraries. Results will be replied to the user on the web interface.

Obtained:

Rather than having it executed from Ignis's GUI, I added a section where users can upload scan results into the vector databases.

Automated Database Updates

Expected:

Weekly scans are conducted and stored using VMs simulating a network environment of an organization.

Obtained:

Attempted to set up the environment but due to Nessus licensing issues I failed to set up Nessus scanners on Virtual Machines.

Targeted Remediation

Expected:

Provide recommendations based on an existing knowledge base.

Obtained:

The vector database allows Ignis to also store knowledge bases for remediation. However, Ignis often pulls remediations from the scan reports themselves unless specified not to do so. (uses less resources possibly.)

Incident Triaging

Expected:

Assign vulnerabilities to the appropriate response team through email.

Obtained:

There is an email section where Ignis can create the email body based on chat history. I also added a triaging system where Ignis parses the chat history, outputs the main vulnerability discussed, and assigns it to a remediation team. There is a page dedicated to displaying live vulnerabilities and assigned teams.

Notable application Differences between start and end of development

Large Language Model

Initial LLM: Llama 2

At the beginning of the development, Ignis utilized Llama 2, a locally hosted LLM, to power its natural language processing capabilities.

Transition to GPT 3.5 Turbo

As the development progressed, the decision was made to switch from Llama 2 to GPT 3.5 Turbo, a cloud based LLM. GPT 3.5 Turbo offered enhanced performance, improved language understanding, and a more extensive knowledge base. The transition to a cloud based LLM also eliminated the need for local hosting, reducing infrastructure complexity and maintenance efforts.

Web Framework

Initial Framework: Flask

During the early stages of development, Ignis was built using the Flask web framework.

Migration to Streamlit

As the project evolved, the decision was made to switch from Flask to Streamlit. Streamlit offered a more user-friendly and intuitive framework for building interactive web applications, making it easier to create and customize the chatbot's user interface. The migration to Streamlit streamlined the development process and enhanced the overall user experience of Ignis.

Triaging Mechanism

Initial Approach: Email-based Triaging

In the initial version of Ignis, the triaging mechanism relied on email communication.

Enhanced Triaging with Dedicated Database

As the application matured, a unique incident triaging mechanism was implemented, supported by its own dedicated database. This new triaging system allowed for more efficient and organized handling of user queries and incidents. The dedicated database enabled better tracking, categorization, and prioritization of incidents, improving the overall effectiveness of the chatbot's triaging capabilities.

Scan Report Format

Initial Format: CSV

In the early stages of development, Ignis generated scan reports in the CSV (Comma Separated Values) format. CSV provided a simple and widely compatible format for storing and sharing scan reports.

Transition to PDF

PDF offered a more professional and visually appealing format for presenting scan results. The transition to PDF improved the readability and usability of the scan reports, making them more accessible and easier to share with future stakeholders.

Databases

Initial Database: FAISS

Initially, Ignis utilized FAISS (Facebook AI Similarity Search) as its vector database solution. FAISS provided basic similarity search capabilities, enabling the chatbot to retrieve relevant information based on user queries.

Migration to ChromaDB

ChromaDB offered enhanced performance, scalability, and flexibility compared to FAISS. The migration to ChromaDB improved the chatbot's ability to handle larger datasets and complex queries, ensuring faster and more accurate responses.

Learning Outcomes

During the project, several key learnings were acquired:

1. **Vulnerability Management Process**: Gained an understanding of how vulnerability management processes are designed and executed. This involves identifying, classifying, remediating, and validating vulnerabilities in software.
2. **App Building that fits real world scenarios**: Applications must be made with the goal always in mind during development. Having the application tested with cybersecurity experts ensures the application benefits the target audience, in this case cybersecurity and TVM (Threat and Vulnerability Management) personnel.
3. **AI Reasoning Capabilities and Limitations**: The project exposed limitations in AI reasoning, showing that often the capabilities of AI in decision-making can be overestimated and unreliable. It also depends on the language models used. High Tier models such as GPT 4 or Claude 3 perform better than previous generation models such as GPT 3.5 or Llama 2 even.
4. **Integration of AI with Data Management Systems**: Successfully implemented LangChain to integrate AI into the data management systems, despite its arbitrary naming conventions and poor library management. This overall allowed for more efficient processing and analysis of data through AI-driven insights.
5. **Importance of Managing Credentials**: Recognized the critical importance of securely managing credentials to prevent unauthorized access and ensure the integrity of the system.
6. **Vector Databases and Embeddings**: Explored the capabilities of vector databases and learned how embeddings can be effectively utilized to improve the accuracy and performance of AI applications.
7. **AI Biases and Fine**-Tuning: LLMs are predisposed to follow certain formats and self-biases which may affect overall output. We must use prompts and fine-tuning through large data sampling to ensure LLMs produce output fitting the given task.

Improvements for Future Initiatives

If I were to redo project development, there would be several steps to adjust:

- **Better balance on Web Application Building**: Initially, there was a substantial focus on developing the backend portion of the project. We must put emphasis on ease of use, which increases the likelihood of my application being chosen over other solutions.
- **Dataset Enhancement**: Would invest more effort into fine-tuning existing datasets and creating enriched sample data which could lead to more robust AI models and simulations.
- **Increased Focus on Incident Triaging**: Recognizing the importance of this aspect, more resources and time would be allocated to develop effective strategies for incident triaging to manage potential threats and vulnerabilities more promptly. Often vulnerabilities would take a long time to resolve and be validated due to poor management. I would implement a more automated system to assign vulnerabilities to remediation teams and monitor their remediation progress.
- **Host vector database on MongoDB or Pinecone**: There are certain pros and cons on having a cloud based commercial database or locally hosted database in the organization. Cloud database providers would reduce workload to setup and manage database updates and security patches.
- **Set up proper scanning environments**: Other security companies such as Tenable or Qualys can offer to set up scanners for organizations. While my application is more focused on assessing and remediating vulnerabilities, having a proper set up of scanners would show off more of Ignis' capability to handle large datasets and history databanks.
- **Proprietary Scanners:** Ignis is built around being able to process scan reports from various software (Nessus, Nmap, Zap). We can fine-tune and edit prompts to fit with a specific scan report format to improve data parsing and chat output from OpenAI API.
- **QoL Features:** Voice recognition and image processing (screenshots of vulnerabilities) to improve quality of life performance of the application.

Conclusion:

Large Language Models (LLMs) are now widely used in many sectors, showing their big influence. My current project is a great example of how we can make managing threats and vulnerabilities much better.

One of the many struggles TVM members go through is assessing the vulnerability properly. Members usually cross check with historical reports of hosts to know more about the vulnerability. Assigning it to the correct team can be a hassle as well.

Through Ignis, that aims on improving user-friendliness for cybersecurity interaction, speed on familiarizing with the network environment and linking directly with a scanning database, this project makes security checks easier and more effective. It illustrates how LLMs can be used not just for handling large data but also for improving cybersecurity as our world grows more connected.

I hope through Ignis's existence as an easy to use, text-based interaction security platform, we've made various vulnerability management processes smoother, which helps perform security assessments more efficiently. Using LLMs speeds up how fast we process security threats and strengthens our IT systems against complex cyber threats. Ultimately, implementing AI capabilities highlights a shift towards more secure, effective, and adaptable IT security space, and is a key advancement in managing future threats and security practices.

Appendix

Declaration of Plagiarism

I declare that this document and any previous submissions by myself is of my own work unless mentioned and acknowledged otherwise. Any external sources such as diagrams, quotes, tables etc. are all cited properly. This includes software and other electronic media intellectual property may reside. I am aware and understand that failure to follow any of the Institute's regulations governing plagiarism constitutes a serious offense.

Student Name: Khairul Amirin Bin Syahrean

Student Number: C00265680

Signature Date: 19/04/2024

Supervisor: Richard Butler

Institution: South Eash Technological University

Signed: ______________________________