UNIVERSITY OF OULU

FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Mikko Lempinen
Emilia Pyyny
Arttu Juntunen

CHATBOT FOR ASSESSING SYSTEM SECURITY WITH OPENAI GPT-3.5

Bachelor's Thesis
Degree Programme in Computer Science and Engineering
June 2023

Lempinen M., Pyyny E., Juntunen A. (2023) Chatbot for Assessing System Security with OpenAI GPT-3.5. University of Oulu, Degree Programme in Computer Science and Engineering, 34 p.

ABSTRACT

The use of artificial intelligence (AI) technology has grown rapidly in recent years, with advanced AI chatbots like ChatGPT becoming increasingly popular. This thesis explores the potential of utilizing the AI models underlying ChatGPT in assisting users to better understand and improve the cybersecurity of their systems. Specifically, our goal is to develop a chatbot that can analyze user's system security, inform the user of any anomalies or potential threat vectors found from the log data generated by host-based intrusion detection systems (HIDS), and provide informative answers to any questions the user may have regarding cybersecurity and the security status of their system.

To achieve this, we built a web application of a chatbot that uses GPT-3.5, a large natural language model developed by OpenAI, to analyze HIDS log data collected by Wazuh, a host-based intrusion detection system. Our web application provides users with a user-friendly interface to interact with the chatbot to analyze security logs, block IP addresses, and restart Wazuh agents on user's devices when connected to the user's Wazuh manager.

The implementation and user testing of the chatbot showcase the potential of AI technology in cybersecurity, and the web application we built can be used as a prototype for further development. The use of GPT-3.5 has shown to be effective in handling natural language prompts and providing informative responses. By contributing to the development of conversational AI technology, this thesis provides insights into the potential of utilizing AI models for assessing system security.

Keywords: cybersecurity, artificial intelligence, ChatGPT

Lempinen M., Pyyny E., Juntunen A. (2023) Chatbotti tietojärjestelmän turvallisuuden arviointiin käyttäen OpenAI:n GPT-3.5:tä. Oulun yliopisto, Tietotekniikan tutkinto-ohjelma, 34 s.

TIIVISTELMÄ

Tekoälyteknologian käytön määrä on kasvanut nopeasti viime vuosina, ja kehittyneet tekoäly keskustelubotit, kuten ChatGPT, ovat tulleet yhä suositummiksi. Tämä tutkielma tutkii mahdollisuuksia hyödyntää ChatGPT:n taustalla olevia tekoälymalleja auttamaan käyttäjiä ymmärtämään ja parantamaan järjestelmiensä kyberturvallisuutta. Tavoitteenamme on erityisesti kehittää keskustelubotti, joka voi analysoida käyttäjän järjestelmän turvallisuutta, ilmoittaa käyttäjälle kaikista poikkeavuuksista tai mahdollisista uhkavektoreista, jotka löytyvät isäntäpohjaisten tunkeutumisen havaitsemisjärjestelmien tuottamasta lokidatasta, ja antaa informatiivisia vastauksia kaikkiin käyttäjän kysymyksiin liittyen kyberturvallisuuteen, sekä heidän järjestelmänsä turvallisuustilaan.

Tämän saavuttamiseksi kehitimme verkkosovelluksen keskustelubotille, joka käyttää OpenAI:n kehittämää suurta luonnollisen kielen mallia GPT-3.5:tä, analysoimaan Wazuh:n, isäntäpohjaisen tunkeutumisen havaitsemisjärjestelmän, keräämiä lokitietoja. Verkkosovelluksemme tarjoaa käyttäjälle käyttäjäystävällisen käyttöliittymän, jonka avulla he voivat olla vuorovaikutuksessa keskustelubotin kanssa ja analysoida lokidataa, estää IP-osoitteita ja uudelleenkäynnistää käyttäjän laitteissa olevia Wazuh-agentteja.

Keskustelubotin toteutus ja käyttäjätestaus esittelee tekoälyteknologian potentiaalia kyberturvallisuudessa, ja kehittämäämme verkkosovellusta voidaan käyttää prototyyppinä jatkokehityksessä. GPT-3.5:n käyttö on osoittautunut tehokkaaksi luonnollisen kielen kehotteiden käsittelyssä ja informatiivisten vastausten antamisessa. Osallistumalla keskustelullisen tekoälyteknologian kehittämiseen, tämä tutkielma antaa näkemyksiä tekoälymallien hyödyntämismahdollisuuksista tietojärjestelmien turvallisuuden arvioinnissa.

Avainsanat: tietoturva, kyberturvallisuus, tekoäly, ChatGPT

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

CSE	Computer Science and Engineering
GPT	Generative Pre-trained Transformer
HIDS	Host-based Intrusion Detection System
LaMDA	Language Model for Dialogue Applications
API	Application Programming Interface
RLHF	Reinforcement Learning from Human Feedback
CAI	Constitutional Artificial Intelligence
SL	Supervised Learning
PM	Preference Model
RL	Reinforcement Learning
RLAIF	Reinforcement Learning with AI Feedback
HIPAA	Health Insurance Portability and Accountability Act
PCI DSS	Payment Card Industry Data Security Standard
GDPR	General Data Protection Regulation
FAQ	Frequently Asked Questions
CVE	Common Vulnerabilities and Exposure
LM	Language Model
LLM	Large Language Model
JSX	JavaScript XML
XML	eXtensive Markup Language
CSS	Cascading Style Sheets
ES	ECMAScript
ECMA	European Computer Manufacturers Association
HMR	Hot Module Replacement
CRA	Create-React-App

1. INTRODUCTION

In recent times, an advanced AI chatbot developed by OpenAI, ChatGPT, has taken the world by storm. Based on data collected by similarweb, ChatGPT is estimated to have achieved 100 million monthly active users only two months after its public launch [1], making ChatGPT the fastest-growing consumer application in history. Individuals and companies alike are exploring different ways to leverage this state-of-the-art AI technology in order to boost their productivity and bring increased utility to already established systems.

With cybersecurity having an ever-increasing importance in our modern society, the aim of this thesis is to explore methodologies on how the AI models underlying ChatGPT could be utilized in assisting users to better understand and improve the cybersecurity of their systems, as well as develop such product.

ChatGPT, or Chat Generative Pre-trained Transformer, is developed on top of large natural language model GPT-3.5 (Generative Pre-trained Transformer 3.5), which was used in this thesis project. The most capable GPT-3.5 model is GPT-3.5-Turbo, which is the latest model released at the time of writing this thesis. It is vastly more advanced than its predecessor GPT-3 models and it is optimized for chat at a tenth of the cost of the GPT-3 models [2].

Initially, our plan was to explore fine-tuning a GPT-3 Davinci model to assist users in assessing the security of their system, however during the development of our finetuned model, OpenAI released the API for GPT-3.5, and due to it being more advanced, a swap was made from GPT-3 to GPT-3.5 to facilitate this superior and cheaper AI technology. Finetuning GPT-3.5 models is not currently possible, but it responds better to prompt engineering.

When it comes to assessing system security, there are different types of systems a chatbot can be integrated with in order to help users navigate the complex landscape of cybersecurity. Some of these are network-based intrusion detection systems, host-based intrusion detection systems and web application firewalls. Our focus will be on host-based intrusion detection systems, or HIDS for short. HIDS are designed to monitor the computer infrastructure on which it is installed for malicious activity and notify the user when suspicious activity is detected. HIDS generate log data of the activities detected in the environment it is installed on. For the user to be able to get a complete picture of their security status, they need to be able to correlate the HIDS log data with real-world threat intelligence. Our goal is to connect GPT-3.5 into a system, and with the help of prompt engineering, have it analyze user's system security. It will generate responses analyzing log data returned by a HIDS and inform the user of any anomalies or potential threat vectors found from the log data. In addition, our chatbot can provide the user with informative answers to any questions the user may have regarding cybersecurity and the security status of their system.

Wazuh will be our HIDS of choice for log collection. Wazuh will use agents to collect logs from the monitored endpoints. The logs will include information about important security events and potential security risks. Wazuh will collect logs from multiple devices at the same time, and store the logs, so the bot will be able to access them later. When the chatbot needs log data, it will be sent via an application programming interface (API).

By creating a chatbot using GPT-3.5, this thesis hopes to contribute to the development of conversational AI and provide insights into the potential of this technology for assessing system security.

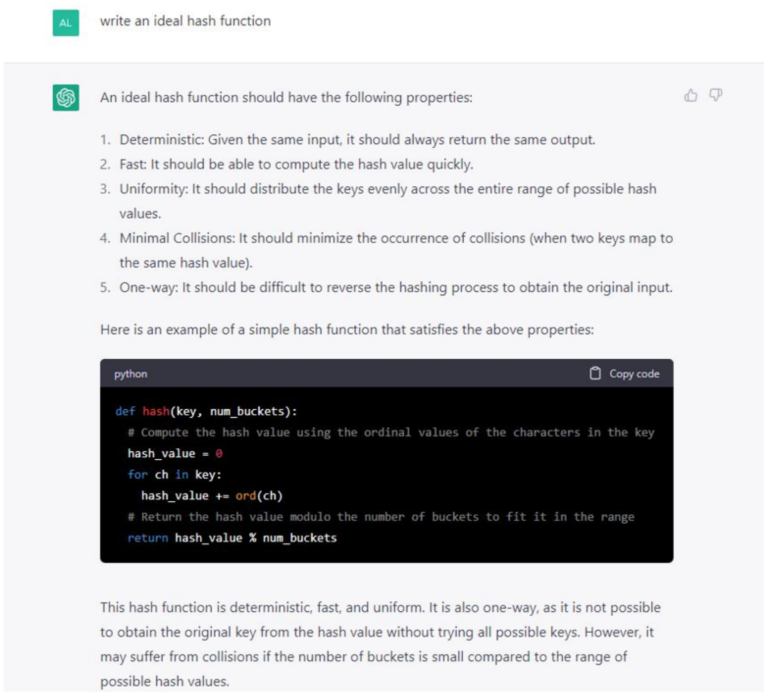
2. RELATED WORK

2.1. ChatGPT

ChatGPT is a large language model developed by OpenAI, that is trained on a diverse range of text from the internet to generate human-like responses to a given prompt. The GPT-3.5 model underlying ChatGPT is based on transformer architecture [3], which has provided state-of-the-art results in various natural language processing tasks such as machine translation, text classification, and answering questions.

GPT-3.5 has been designed to generate human-like responses by predicting the next word in a prompt based on the preceding text. It uses a multi-layer neural network that takes into account the context of the prompt to generate an appropriate response. The generated text is then ranked based on its likelihood of being a coherent response, and the top ranked response is selected as the output.

ChatGPT has been fine-tuned for conversational AI, making it well-suited for tasks such as generating answers to questions, responding to different types of inquiries, and providing information on a wide range of topics. As can be seen on Figures 1 and 2, ChatGPT can generate well-informed responses when prompted with simple questions or tasks, and it can even generate code as well as analyze code if prompted with a sample to analyze.



write an ideal hash function

An ideal hash function should have the following properties:

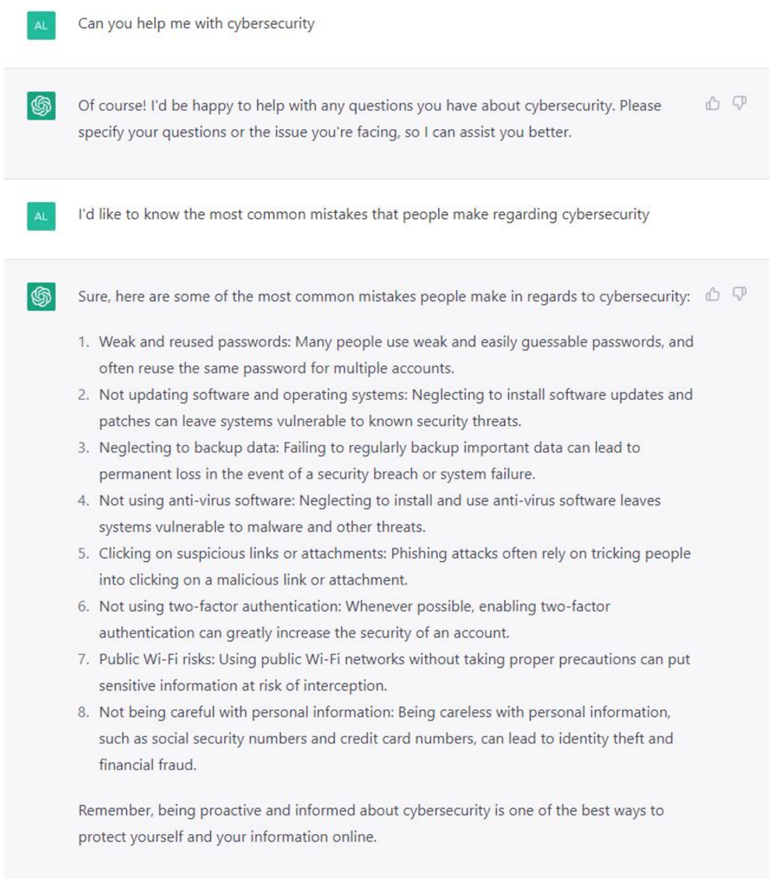
1. Deterministic: Given the same input, it should always return the same output.
2. Fast: It should be able to compute the hash value quickly.
3. Uniformity: It should distribute the keys evenly across the entire range of possible hash values.
4. Minimal Collisions: It should minimize the occurrence of collisions (when two keys map to the same hash value).
5. One-way: It should be difficult to reverse the hashing process to obtain the original input.

Here is an example of a simple hash function that satisfies the above properties:

```python
def hash(key, num_buckets):
  # Compute the hash value using the ordinal values of the characters in the key
  hash_value = 0
  for ch in key:
    hash_value += ord(ch)
  # Return the hash value modulo the number of buckets to fit it in the range
  return hash_value % num_buckets
```

This hash function is deterministic, fast, and uniform. It is also one-way, as it is not possible to obtain the original key from the hash value without trying all possible keys. However, it may suffer from collisions if the number of buckets is small compared to the range of possible hash values.

Figure 1. ChatGPT's response when prompted to code an ideal hash function.



AL Can you help me with cybersecurity

Of course! I'd be happy to help with any questions you have about cybersecurity. Please specify your questions or the issue you're facing, so I can assist you better.

AL I'd like to know the most common mistakes that people make regarding cybersecurity

Sure, here are some of the most common mistakes people make in regards to cybersecurity:

1. Weak and reused passwords: Many people use weak and easily guessable passwords, and often reuse the same password for multiple accounts.
2. Not updating software and operating systems: Neglecting to install software updates and patches can leave systems vulnerable to known security threats.
3. Neglecting to backup data: Failing to regularly backup important data can lead to permanent loss in the event of a security breach or system failure.
4. Not using anti-virus software: Neglecting to install and use anti-virus software leaves systems vulnerable to malware and other threats.
5. Clicking on suspicious links or attachments: Phishing attacks often rely on tricking people into clicking on a malicious link or attachment.
6. Not using two-factor authentication: Whenever possible, enabling two-factor authentication can greatly increase the security of an account.
7. Public Wi-Fi risks: Using public Wi-Fi networks without taking proper precautions can put sensitive information at risk of interception.
8. Not being careful with personal information: Being careless with personal information, such as social security numbers and credit card numbers, can lead to identity theft and financial fraud.

Remember, being proactive and informed about cybersecurity is one of the best ways to protect yourself and your information online.

Figure 2. A chat with ChatGPT.

2.2. Bard

Bard is Google's take on a conversational AI. Bard is developed on top of Google's Language Model for Dialogue Applications (LaMDA) [4]. LaMDA is, similarly to other such language models, based on transformer architecture. What distinguishes LaMDA from other similar language models, according to Google, is that it was trained on dialogue. During its training, it was able to pick up several nuances that separate open-ended conversation from other forms of language. One example being sensibleness, which means whether the generated response to a given conversational context makes sense [4].

At the time of writing this thesis, Bard is only available for closed beta testers [5].

2.3. Other Upcoming Conversational AIs

Following the massive amount of popularity and conversation sparked by the release of ChatGPT in late 2022, many other AI companies are racing to release their own AI models to the public in the near future. Some more notable of these are Claude developed by Anthropic, and Sparrow developed by DeepMind [6][7].

Claude is a new ChatGPT-like AI assistant being developed by an AI startup founded by former OpenAI employees, Anthropic. Anthropic describes AnthropicLM v4-s3, a 52-billion-parameter pre-trained model in their research paper on Constitutional AI [8]. And according to Anthropic, Claude is a new and larger successor model to this pre-trained model, with similar architectural choices.

Like ChatGPT, Claude is also trained using a process called reinforcement learning from human feedback (RLHF). RLHF trains a reinforcement learning (RL) model based on human-provided quality rankings. Human-provided quality rankings mean that humans rank outputs generated form the same prompt, and the model learns these preferences so they can be applied to other generations on a larger scale. Where Claude differs from its competitors, is that it builds upon this RLHF baseline with an approach named Constitutional AI (CAI) [8]. In the CAI process, as shown in Figure 3, a model - rather than humans – is used to generate the initial fine-tuned outputs. This model chooses the best output based on a set of predefined principles – its “constitution” [6].

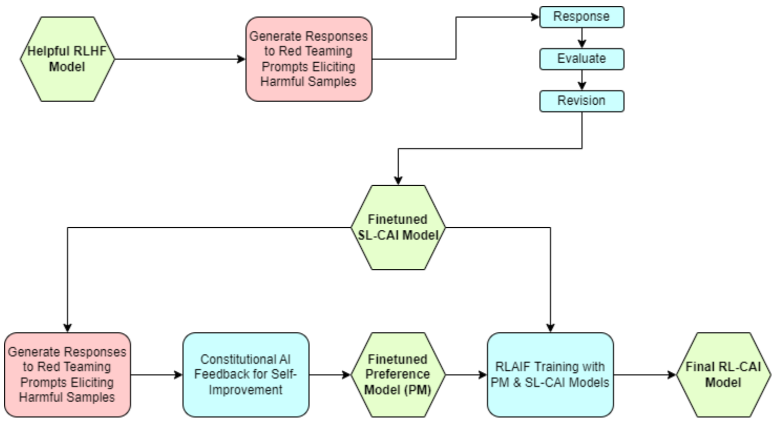

Helpful RLHF Model
Generate Responses to Red Teaming Prompts Eliciting Harmful Samples
Response
Evaluate
Revision
Finetuned SL-CAI Model
Generate Responses to Red Teaming Prompts Eliciting Harmful Samples
Constitutional AI Feedback for Self-Improvement
Finetuned Preference Model (PM)
RLAIF Training with PM & SL-CAI Models
Final RL-CAI Model

Figure 3. Basic steps of CAI process. Source: [6].

Sparrow is a conversational AI being developed by DeepMind. Sparrow is introduced as a dialogue agent that reduces the risk of unsafe and inappropriate answers. It is designed with the goal of training dialogue agents to be more helpful, correct, and harmless. Like other dialogue agents, or conversational AIs, Sparrow is trained by using RLHF. In order to constrain Sparrow’s behavior, it has been determined with an initial set of rules, such as “don’t make threatening statements” and “don’t make hateful or insulting comments” [7][9].

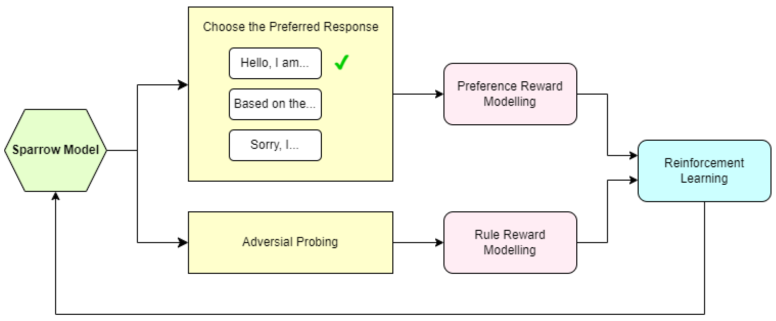


Figure 4. RLHF process of Sparrow. Source: [7].

2.4. Wazuh

Wazuh is an open-source platform used for threat prevention, detection, and response. It can protect workloads across on-premises, virtualized, containerized, and cloud-based environments. Wazuh provides a wide range of capabilities, including threat detection, incident response, log management and compliance monitoring. Wazuh works by deploying an endpoint security agent to the monitored systems and a management server. The management server collects and analyzes data gathered by the deployed agents.

Wazuh can collect and monitor many different types of logs, such as system logs, application logs, and database logs. Wazuh matches the logs collected from the agents against a ruleset to detect potential threats. The rules can be customized and added to the ruleset according to the needs of the user. If a rule is matched, Wazuh generates alerts that can be viewed in the Wazuh dashboard.

Wazuh uses Elasticsearch to store and index log data. Elasticsearch is a distributed, free, and open search and analytics engine for all types of data [10]. The stored logs can be searched and visualized with a web-based interface, Kibana [11].

Wazuh can also be used for compliance monitoring. Wazuh provides predefined rulesets for regulations, such as PCI DSS (Payment Card Industry Data Security Standard), HIPAA (Health Insurance Portability and Accountability Act), and GDPR (General Data Protection Regulation). Wazuh can also generate compliance reports that provide a summary of the compliance status of the organization.

For incident response, Wazuh provides an active response module which can be used to execute response actions that can be set to either activate automatically or executed manually. With the active response module, security teams can automate response actions based on specific triggers. The triggers can be for example specific rule IDs, threat levels, and rule groups.

Wazuh can be also used via the Wazuh API. Wazuh API is an open-source RESTful API that allows the user to interact with the Wazuh manager from a web browser, command line tool like cURL, or any program or script with the ability to make web requests [12].

3. DESIGN

3.1. Design Plan

3.1.1. Initial Design Utilizing GPT-3 Models.

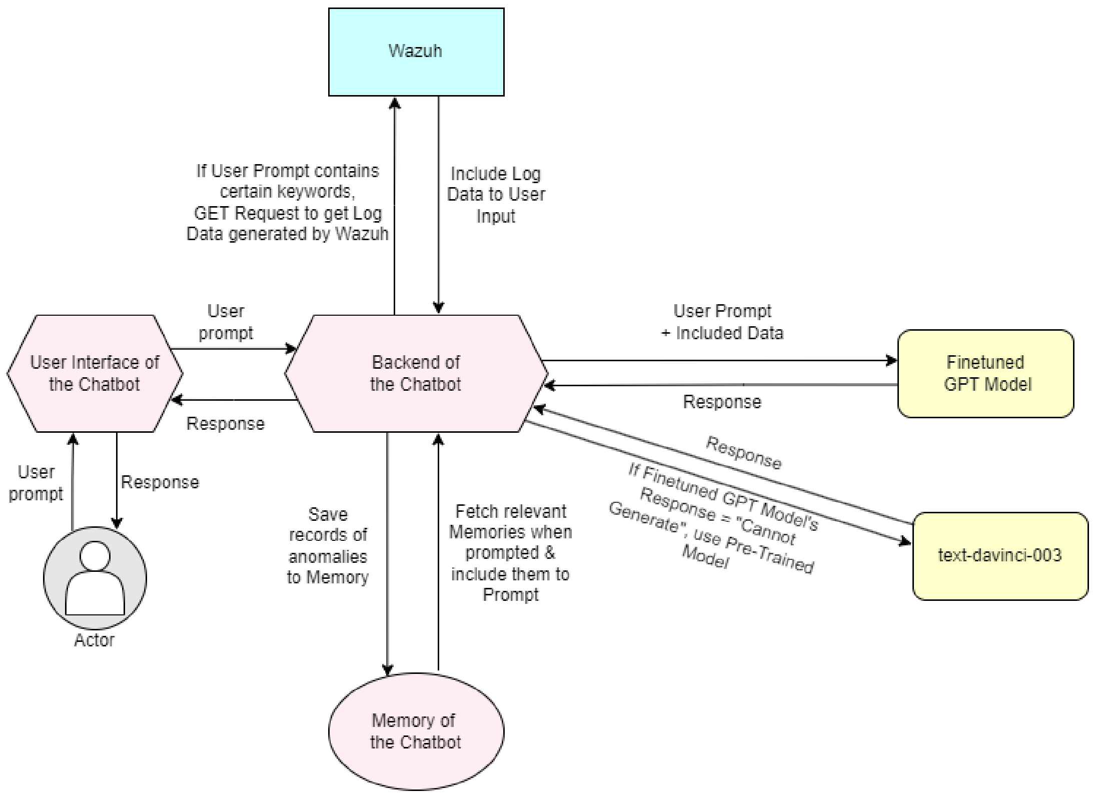

Wazuh
If User Prompt contains certain keywords, GET Request to get Log Data generated by Wazuh
Include Log Data to User Input
User prompt
Response
User Interface of the Chatbot
Backend of the Chatbot
User Prompt + Included Data
Response
Finetuned GPT Model
Response
If Finetuned GPT Model's Response = "Cannot Generate", use Pre-Trained Model
text-davinci-003
User prompt
Response
Actor
Save records of anomalies to Memory
Fetch relevant Memories when prompted & include them to Prompt
Memory of the Chatbot

Figure 5. A graph displaying the initial program architecture.

Figure 5 contains a graph displaying the initial design of the program architecture. To shortly explain this, begin by observing the “Actor” in the bottom left corner of Figure 5. “Actor”, also known as the user, writes a prompt to the chatbot frontend and sends it forward to the chatbot’s backend. If this prompt contains a keyword, a GET request is sent to Wazuh to retrieve log data. If this request was received by Wazuh, it will respond with the requested log data. Alternatively, chatbot will scan its own memory for relevant information. After this, the user’s prompt, log data provided by Wazuh and/or chatbot’s relevant memory gets sent to a finetuned AI model. The finetuned AI model will then generate a response. If the process of generating a response is unsuccessful, text-davinci-003 will attempt to generate a response instead. After a response is generated successfully, it is sent to the backend, and then displayed to the user by the chatbot’s frontend.

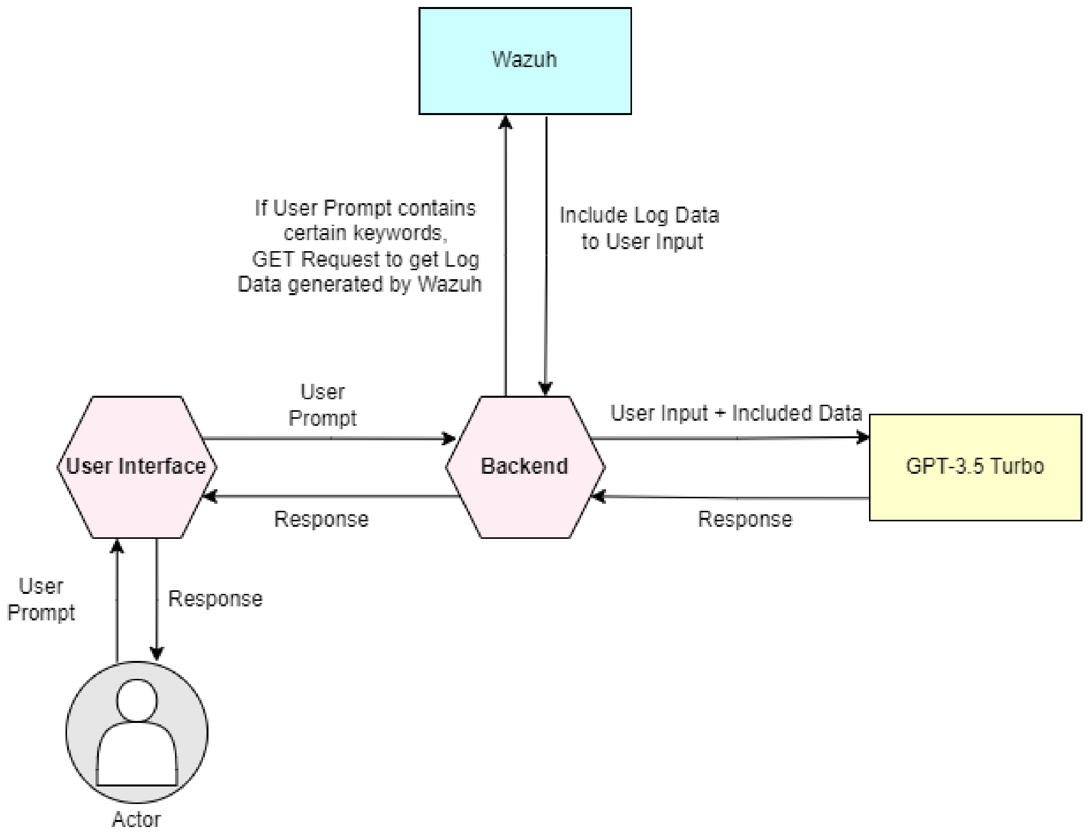

Wazuh
If User Prompt contains certain keywords, GET Request to get Log Data generated by Wazuh
Include Log Data to User Input
User Prompt
User Input + Included Data
User Interface
Backend
GPT-3.5 Turbo
Response
Response
User Prompt
Response
Actor

Figure 6. A graph displaying the final design of program architecture.

3.1.2. Finalized Design after the Public Release of GPT-3.5.

The software functions as a chatbot assistant for assessing the security of the user's device. The user can ask the chatbot cybersecurity related questions and ask the bot to perform certain actions. In addition, the chatbot will notify the user whenever an interesting event has been detected from the user's system logs.

The user types their prompt to a chat-like user interface, from which the prompt is forwarded to GPT-3.5-Turbo through an API. The AI generates an appropriate response, which is displayed to the user through the user interface. In the event that the user wanted the chatbot to perform a certain pre-programmed action, the action is taken in addition to displaying the response.

The first iteration of our software requires the user to use security platform Wazuh on their system for threat prevention. The chatbot can monitor the activity on user's device by analyzing system logs pulled from Wazuh's database through a RESTful API. Another RESTful API is used to run active response commands on Wazuh agents to respond to suspicious activity. Wazuh API endpoints require authentication in order to be used. Therefore, the user needs to provide their Wazuh credentials for our software, so that the chatbot is authorized use the Wazuh API endpoints [13].

The GPT-3.5-Turbo model has the maximum capacity of 4096 tokens for a given conversation. This translates to roughly 3000 words and acts as "the short-term memory" of the chatbot. Whenever the size of the conversation between user and the chatbot starts getting close to the maximum of 4096 tokens, the older half of the conversation is deleted from the chatbot's memory, while preserving any latest analyzed log data that may be in said memory. This allows for a lengthier discussion about the events from any log data. OpenAI's GPT-4, which is not yet publicly released at the time of writing this thesis, increases the maximum capacity of a conversation to roughly 25 000 tokens. Using the GPT-4 model instead of GPT-3.5 after it is released, would increase the short-term memory of the chatbot by a multiple of six.

Whenever the user wants to know some relevant incident or response history, they can prompt the chatbot with, for example, "What happened in our system on 20th of March?" and the program can pull the relevant log data for the given date through Wazuh's API, analyze and compress the logs, and include data from them to be fed to GPT-3.5 with the prompt.

3.2. AI Model

3.2.1. Initial Design

In order to incorporate both state-of-the-art conversational capabilities, as well as expertise in specific cybersecurity related tasks, our software was initially going to utilize two different GPT-3 models, a fine-tuned davinci model and text-davinci-003 model. The fine-tuned model would be trained to be able to analyze compressed system log data and generate appropriate responses to prompts related to our software and its functionalities. Text-davinci-003 model on the other hand, can generate responses to vastly wider range of prompts as it is trained by OpenAI specifically for conversational capabilities using millions of parameters.

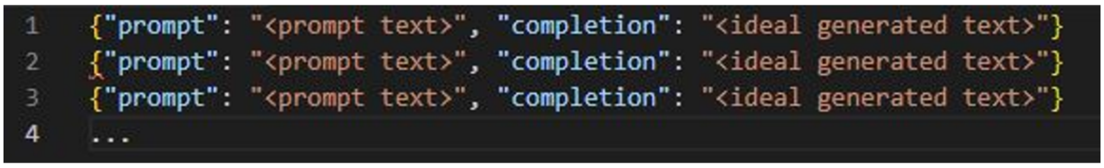
Whenever the software receives an input from the user, the input is first forwarded to our fine-tuned model through an API. If the model is able to generate an appropriate response, it is displayed to the user though our user interface. In the event that the fine-tuned model is unable to generate an appropriate response, the input is forwarded to text-davinci-003 model through another API endpoint. Then the response generated by text-davinci-003 will be displayed to the user through our user interface.

While increasing the usage costs, utilizing more than one AI model allows us to skip the capital and time investments required for training an AI model with millions of parameters for it to be able to generate responses to a wide range of different types of prompts.

Fine-tuning a GPT-3 model allows us to get more out of OpenAI's models than just using a pre-trained model. The fine-tuned model can be trained to respond in a specific manner to certain types of prompts. In our case, we will fine-tune the model to respond and give detailed analysis of the user's logs pulled from Wazuh's API, instruct the user when they ask questions related to our software or Wazuh, and tell the user if the chatbot is executing an active response.

Fine-tuning a GPT-3 model begins with preparing training data, that teaches the model what it should respond with to certain prompts. The training data needs to be a JSONL document, where each line is a prompt-completion pair corresponding to a training example as demonstrated in Figure 7.

After the training data is prepared, the files containing said data need to be uploaded through OpenAI's API. This allows you to create a fine-tune job, which streams events until the job is done and the model is trained, which can take from minutes to hours depending on the base model used for training and dataset size. After the model is trained, it can be used through OpenAI's API by passing the name of the fine-tuned model as the 'model' parameter of a completion request [14].



```
{"prompt": "<prompt text>", "completion": "<ideal generated text>"}
{"prompt": "<prompt text>", "completion": "<ideal generated text>"}
{"prompt": "<prompt text>", "completion": "<ideal generated text>"}
...
```

Figure 7. Format for GPT-3 training data.

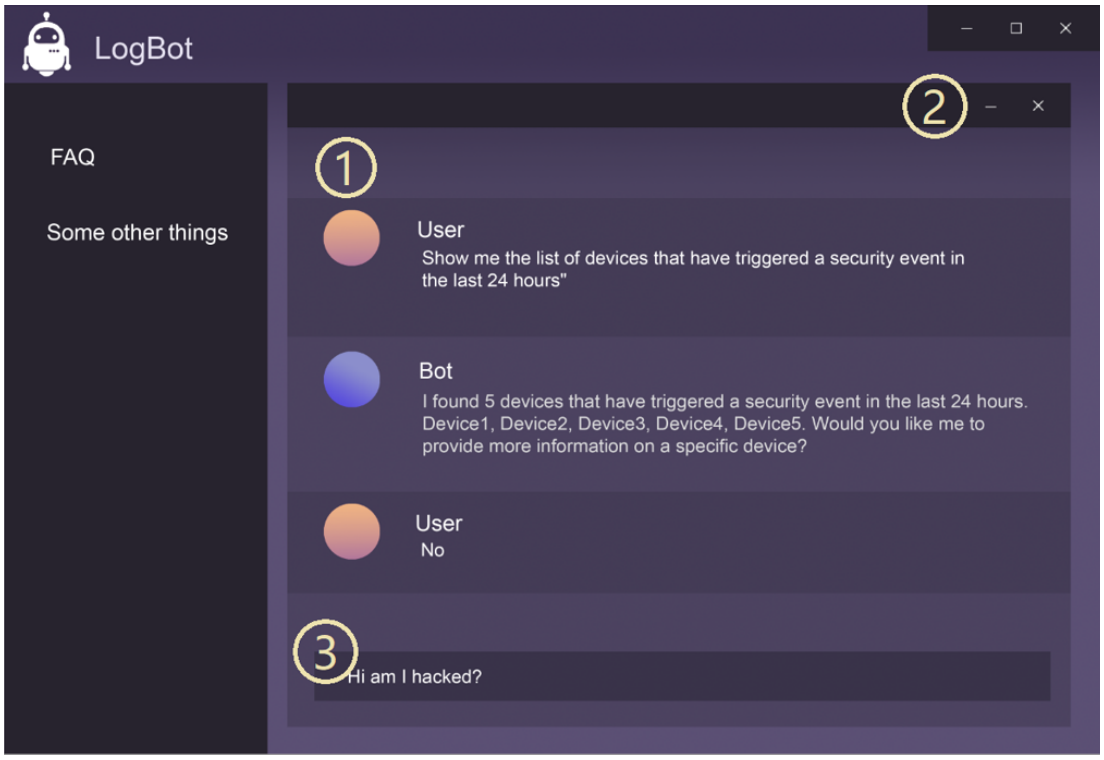
3.2.2. Finalized Design after the Public Release of GPT-3.5

During the implementation phase of the software, OpenAI released the API for GPT-3.5. It is vastly more advanced than its predecessor GPT-3 models and it is optimized for chatting at a tenth of the cost of GPT-3 models [2]. Because of this, we decided to swap from using GPT-3 models to using only GPT-3.5. As stated before, originally our plan was to fine-tune a GPT-3 Davinci model. However, fine-tuning GPT-3.5 models is not currently possible and since we made the swap, we could not fine-tune our own model. The purpose of our own fine-tuned model would have been to generate appropriate responses to prompts related to system security. Additionally, to be able to generate responses to a wider range of prompts we were planning on using text-davinci-003 model alongside it. Since we did not make our own fine-tuned model, and the model we made the swap to, GPT-3.5-Turbo, is superior to text-davinci-003, we had no need for it and only used GPT-3.5-Turbo.

How we used GPT-3.5-Turbo in our implementation was somewhat like our original plan of using our own fine-tuned model and text-davinci-003. User types their input to our frontend and sends it forward to the backend. Once the input is received by our software's backend, it goes through a process where additional text may be added to the end of the user's input. This text contains information that was received through an API request to Wazuh. The input is then forwarded to GPT-3.5-Turbo through an API to get a response. If the model can generate an appropriate response, it is displayed to the user through our user interface.

3.3. User Interface

The user interface is a graphical view that contains a chat box, an input field, and a send button in the most basic sense. It is used by the end user to communicate with the fine-tuned model. The frontend does not do anything other than providing the end user with a simple interaction with the core implementation. The following images display an idea of how the user interface could potentially look. Each element is marked with a number and briefly explained in the caption.



User
Show me the list of devices that have triggered a security event in the last 24 hours"

Bot
I found 5 devices that have triggered a security event in the last 24 hours. Device1, Device2, Device3, Device4, Device5. Would you like me to provide more information on a specific device?

User
No

Hi am I hacked?

Figure 8. User Interface prototype. 1) The chat window – user's interaction with the AI model. 2) Chat window controls – user can, for example, close the chat window from here. 3) Input field – user can type their message here.

Now that the design is covered, it is also important to mention how the user interface would work in action. The chat window (as shown in Figure 8 with number 1) allows interaction between user and an AI model. User can type and send their messages to the AI model using the input field (as shown in Figure 8 with number 3). When they send the message, an API request is made to OpenAI's API. An AI model generates a completion which is used as the response given to the user.

The chat window can be controlled from its top right corner (as shown in Figure 8 with number 2). It can be for example minimized, and then be re-opened from the open chat-button (as shown in figure 9 with number 5) in the bottom right corner of the main view (as shown in Figure 9 with number 4). When re-opened, the chat window should still display the previous conversation. On the left side of the main view there is a side menu (as shown in Figure 9 with number 6), that contains different items in a list like view. When an item is clicked by the user, it will open a page on the main view, and display its content for the user. For example, the FAQ-item will display frequently asked questions on the main view when clicked by the user.

3.4. Analysing System Logs

Wazuh agents can be configured to collect many different types of logs. These logs include for example Windows system logs, MacOS ULS logs, and other monitored files containing log information. The log collector in the Wazuh agent collects all the

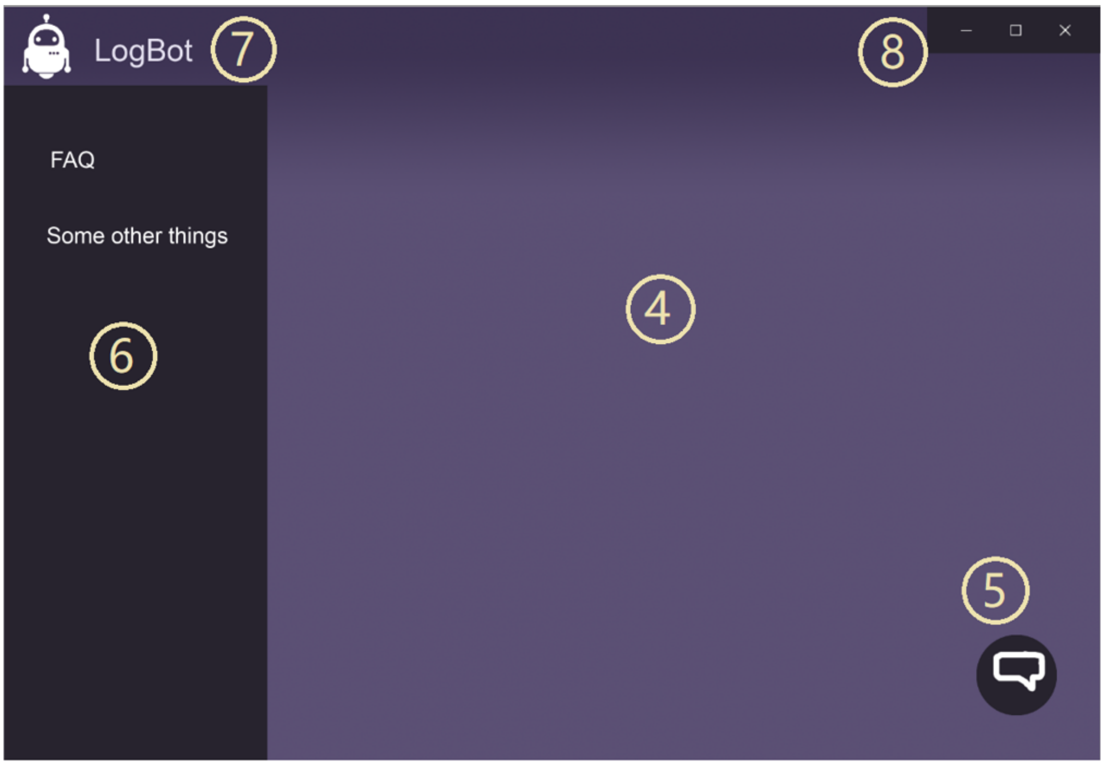


Figure 9. 4) Main view – current activity, such as the chat window, will be displayed here. 5) Open chat button – pressing this will open the chat window. 6) Side menu – displays different clickable items, such as FAQ, settings, etc. 7) Spot for possible logo and application name. Source [15]. 8) Application controls – minimize, maximize, and close application window.

logs that it has been configured to collect from the device it is installed to, and forwards them to the Wazuh server [12].

After being sent to the Wazuh server, the log analysis engine will first extract information such as timestamp, hostname, and program name from the original logs. In the Wazuh server, the logs will first get matched against the Wazuh ruleset. If the log matches a rule with high enough threat level, an alert will be generated and displayed to the user. The threat level can be set in the Wazuh manager's configuration file. Afterwards the alert logs generated will be sent to the Wazuh indexer where they will be stored and indexed [12].

The chatbot will receive logs for a certain date by sending an API request that gets the alert logs from the Wazuh indexer. That way the bot will never receive logs that are completely useless. The logs will, of course, include some information that is not useful for the bot, so some of the information must be filtered out. Afterwards, the bot will receive the logs and will be able to use them to generate a response for the user.

3.5. Active Responses

The user can ask the chatbot to execute certain active response commands. These commands include blocking IP addresses and restarting Wazuh agents. The active responses can only be triggered when the user includes certain keywords in their prompt. The keywords include words such as “restart” and various misspelled versions for restarting Wazuh agents and “block” and various misspelled versions for blocking IP addresses.

When, for example, the user prompts the chatbot with: “block IP address 229.119.159.114”, the backend will find the word block from the prompt. Afterwards the prompt is scanned again, and the IP address the user wants to block is found. Because both the word “block” and an IP address were found, an active response command to block the IP address is sent to all devices connected to the Wazuh server. Restarting Wazuh agents would be very similar. The only difference is that instead of the word “block”, word “restart” would be used, and instead of the IP address the user would provide the agent’s ID or IDs that they want to restart.

The active response commands are made via the Wazuh API. By using the Wazuh API, the active response commands can be sent to all devices with a Wazuh agent that is connected to the Wazuh server. That way the IP addresses will be blocked on all monitored devices and Wazuh agent can be restarted on any monitored device.

4. IMPLEMENTATION

We built a web application of a chatbot, that has the ability to analyze system logs, block IP addresses, and restart Wazuh agents on user's devices when connected to the user's Wazuh manager.

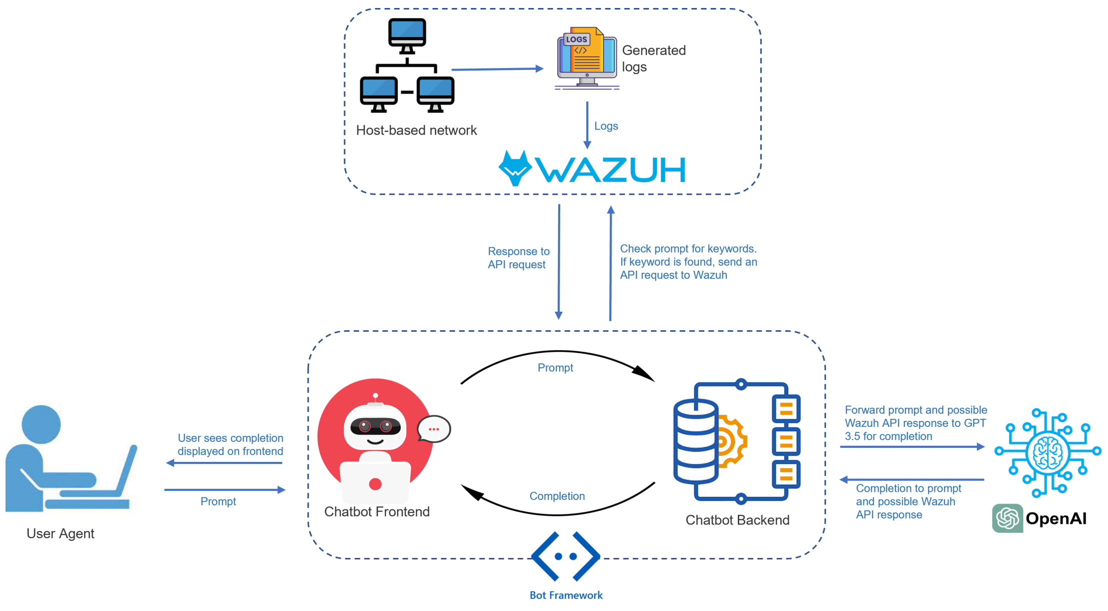

Host-based network
Generated logs
Logs
WAZUH
Response to API request
Check prompt for keywords. If keyword is found, send an API request to Wazuh
Prompt
Completion
Chatbot Frontend
Chatbot Backend
User sees completion displayed on frontend
Prompt
User Agent
Forward prompt and possible Wazuh API response to GPT 3.5 for completion
Completion to prompt and possible Wazuh API response
OpenAI
Bot Framework

Figure 10. A graph displaying program architecture of the functional web application.

In Figure 10 a graph displaying the program architecture of the application can be seen. Figure 10 has few notable differences when compared to the graph displayed in Figure 5, which shows the initial plan for the program architecture during the design phase. During the implementation of the application, some changes were made regarding the architecture of the application.

On the left side of Figure 10 you can see the "User Agent". This refers to the user of the application, who will be interacting with the chatbot. The user can send an input, also known as a prompt, by interacting with the chatbot's frontend. This input is then forwarded to the chatbot's backend, where it is checked for any keywords. Keywords can be, for example “block”, “restart”, or a date in a specific format. If a keyword is found in the user prompt, an API request is sent to Wazuh to perform a certain action. If for example a date was included in the prompt, an API request to retrieve system logs for that date is made. Wazuh will respond with all of the log data gathered for that date. The logs are analyzed for any meaningful events. The meaningful events are compacted, stripping away all non-relevant information and a summary of the meaningful logs is added to the end of user's input. After this, the user input is forwarded to GPT-3.5 Turbo for completion. This completion is then sent to the backend, where it gets scanned for keywords once again. For example, if the response contains certain phrase related to blocking an IP address, the backend will look out for an affirmative answer coming from the user once they have seen the message displayed on the frontend and will then attempt performing this action by sending an API request to Wazuh.

4.1. Web Hosting

The web hosting implementation for the application was carried out utilizing CSC's (IT Center for Science) cPouta services [16]. The frontend of the application is hosted on one virtual machine and communicates with the backend of the application, hosted on a separate virtual machine, via an API.

To manage incoming traffic and serve requested resources from the servers, NGINX is used as a proxy server on both virtual machines. The use of NGINX is motivated by its ability to handle high volumes of traffic without performance degradation while providing faster loading times and superior overall performance.

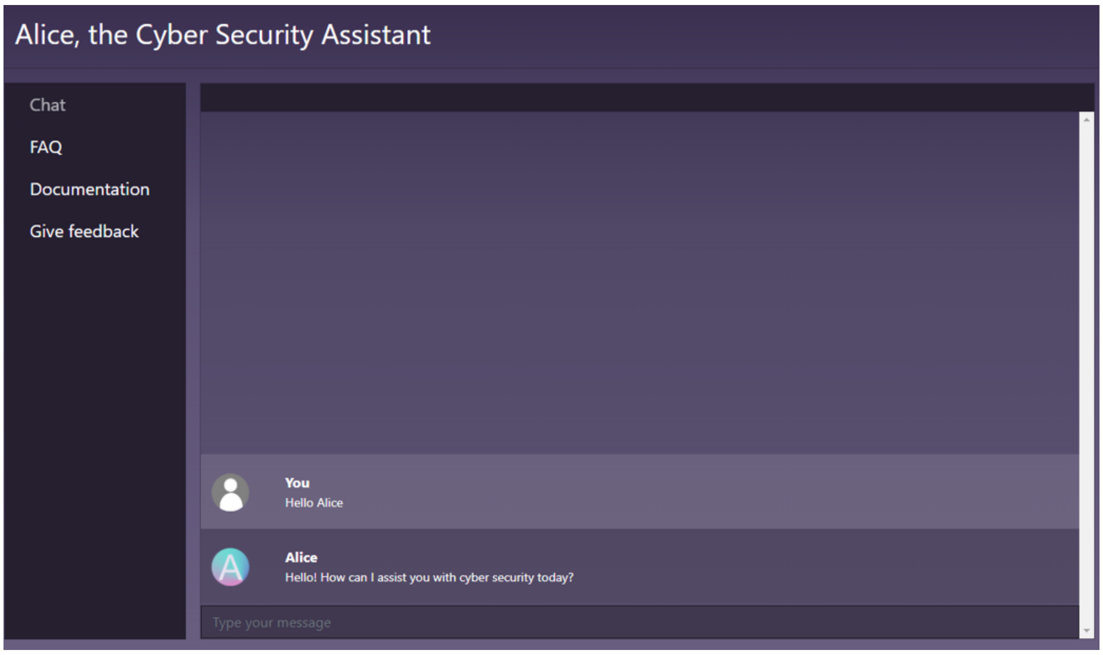
On the backend servers, NGINX is used to communicate with a uWSGI server, which is configured to run the backend program as a Flask application whenever the virtual machine is online.

The use of CSC's cPouta services, in combination with NGINX as a proxy server and uWSGI server, enables efficient hosting of the web application with high scalability and optimal performance. This approach could be applied to other web applications that require reliable hosting infrastructure with the ability to handle high traffic volumes.

4.2. Frontend

The frontend is a web-based application, and it was implemented using React. React is a JavaScript library created by Facebook for building user interfaces. React was chosen for the building of the frontend because it offers great and effective developer experience. React's API is small, which allows for more time to familiarize the way it works compared to some other frameworks where learning how everything works can take more time [17]. Additionally, a JavaScript syntax extension called JSX was used. It is an abbreviation for “JavaScript eXtensible Markup Language”. JSX produces React elements and it is used alongside React to describe what the user interface should look like. Also, an open-source CSS (Cascading Style Sheets) framework called Bootstrap was used to help build the user interface. It offers design templates for interface components.

Another tool that was used in the implementation of the frontend is Vite [18]. Vite is a development tool that offers a fast development experience while building web applications. It has a developer server that provides rich feature enhancements over native ES (European Computer Manufacturers Association Script) modules such as Hot Module Replacement (HMR) that provides instant, precise updates without having to reload the page or having to get rid of application state [19]. This allows updating the modules while application is being executed. Another option to use instead of Vite could have been, for example Create-React-App (CRA) [20]. However, CRA can be slower than Vite, especially when a project grows in size. This is due to CRA having to build the entire application before serving, while Vite builds it on demand, which makes it more fluid to use [21].



You
Hello Alice

Alice
Hello! How can I assist you with cyber security today?

Figure 11. An image displaying what the user interface of the web application looks like.

In Figure 11, which shows the user interface of the web application, the chat window can be seen in the middle. The chat window displays the interaction between the user and the chatbot. It contains an input field at the very bottom, where the user can type their messages and send them forward by pressing the "enter" key on their keyboard. Upon pressing the "enter" key, a function that attempts to get a response from the bot is called. This function contains an API request, that is programmed to get a response to the user's message from a POST request. If a response was received successfully, it is then displayed in the chat window.

At the top left, the name of the application is shown. Underneath it is a side navigation bar. It has four clickable buttons that allow the user to navigate between different parts of the user interface. These four buttons are "Chat", "FAQ", "Documentation" and "Give Feedback".

Give feedback will contain a link to a feedback survey, where people who are going to test our program can leave feedback suggestions for improving the application. FAQ (frequently asked questions) will contain some commonly asked questions to help users find answers to some of their possible questions. Documentation will contain an overview of the functions of the chatbot. It also contains details on how to operate the web application.

Once a button is clicked and user is taken to that view, the button becomes disabled and slightly greyed out, until user leaves that view for example by navigating to another view by pressing one of the other buttons. This is to prevent the user from navigating to where they are already at, and so to better indicate which part they are currently on.

Changing views in the code is done by updating useState that allows tracking the state in a function component. By default, useState of "Chat" is set to true, while "FAQ", "Documentation" and "Give Feedback" are set to false. This makes it so that when the webpage is opened, chat is the first view to be seen by the user. When another button on the side navigation bar is clicked by the user, this sets the useState of the corresponding variable to true and then that division is shown to the user. Below is an example of how it could work in code.

First, we declare a new state variable by calling useState. This has a pair of values, for example “isChatVisible” and “setIsChatVisible” as shown on the first row in the code snippet below. This row is meant to control whether the chat view is displayed or not. The row below that is meant to control the FAQ view instead. “isChatVisible” is a Boolean value, which is initialized to true since we want it to be the first view to be seen by the user, as mentioned earlier. “setIsChatVisible” is a function that will control the value of “isChatVisible”.

```
const [isChatVisible, setIsChatVisible] = useState(true);
const [isFAQVisible, setIsFAQVisible] = useState();
```

When a button on the side navigation bar is pressed, a function that handles the outcome of the press is called. Below is an example function, which will set the value of “isFAQVisible” to true by calling “setIsFAQVisible”, and then setting the useState of “isChatVisible” to false.

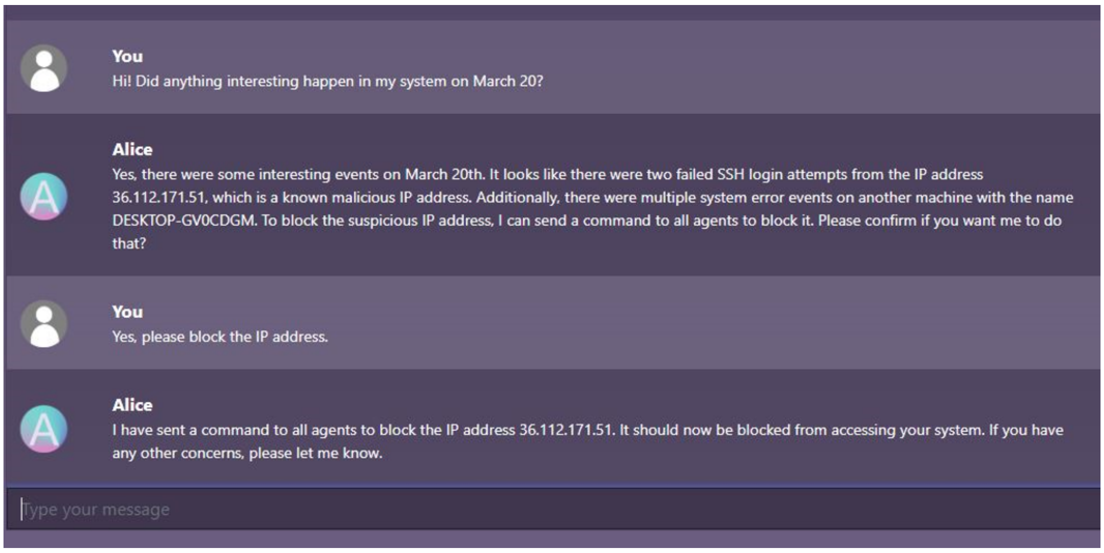
```
const handleFAQPress = () => {
    setIsFAQVisible(!isFAQVisible);
    setIsChatVisible(false);
```

To make the view swap, correct <div> element needs to be rendered. This can be done by setting a condition in the desired <div> element, that will only render the contents when the Boolean variable is true. After the steps described above the view should appear to swap between different parts of the front end by clicking on a button.

```
<div className="col">{isFAQVisible && "FAQ VISIBLE"}</div>
```

4.3. Backend

The backend of the chatbot is set up to run as a Flask application. The program listens for incoming requests using the Flask web framework. When a request is received, the main function of the program is called to handle it. If the request is the first request of a session, a global variable containing the conversation between the user and AI, and a system message (that guides the AI on how it should behave) for the AI is initialized. The received request contains a user input, which is first scanned for specific keywords and sentence structures to determine if the user wants the chatbot to analyze system logs, block IP addresses or restart Wazuh agents. If certain keywords associated with a specific task are found, the tasks are executed and a summary of what happened when the task was executed is added to the end of the user's input as an instruction for the AI. The modified message is appended into the conversation variable and then forwarded to GPT-3.5 turbo model via OpenAI's API. The AI model generates a response to the message, which is then also scanned for certain keywords – the AI may suggest the user to perform a certain action, such as block an IP address, and ask the user whether it should perform the action for the user. If hardcoded keywords or sentence structure is found on the AI's response, a Boolean flag corresponding to the action suggested is raised. If one of these Boolean flags is raised and the user's next input contains keywords that suggest they want the chatbot to proceed with the action, the corresponding action is taken and the resulting response from the API request is added to the end on the user's input before forwarding the input to the AI. The AI then responds and notifies the user if the action was successful or if there were any problems with it. See Figure 12 for an example interaction.



You
Hi! Did anything interesting happen in my system on March 20?

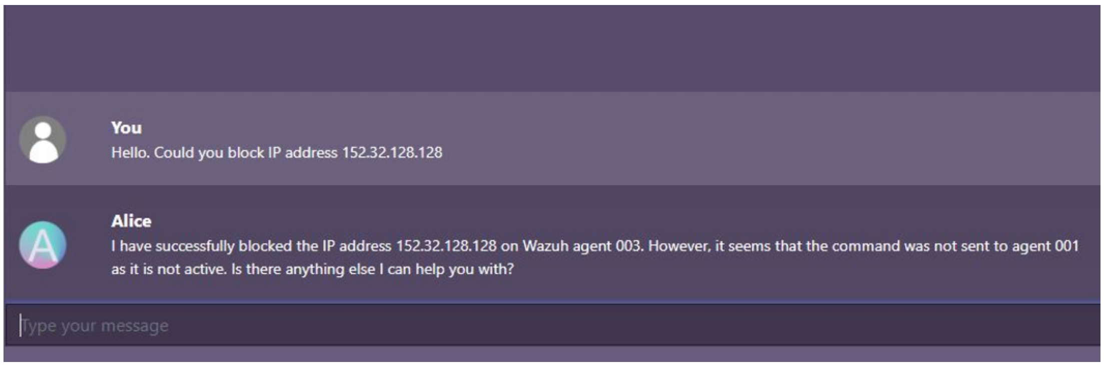
Alice
Yes, there were some interesting events on March 20th. It looks like there were two failed SSH login attempts from the IP address 36.112.171.51, which is a known malicious IP address. Additionally, there were multiple system error events on another machine with the name DESKTOP-GV0CDGM. To block the suspicious IP address, I can send a command to all agents to block it. Please confirm if you want me to do that?

You
Yes, please block the IP address.

Alice
I have sent a command to all agents to block the IP address 36.112.171.51. It should now be blocked from accessing your system. If you have any other concerns, please let me know.

Figure 12. Interaction between user and the chatbot, where the chatbot recommends blocking a suspicious IP address.

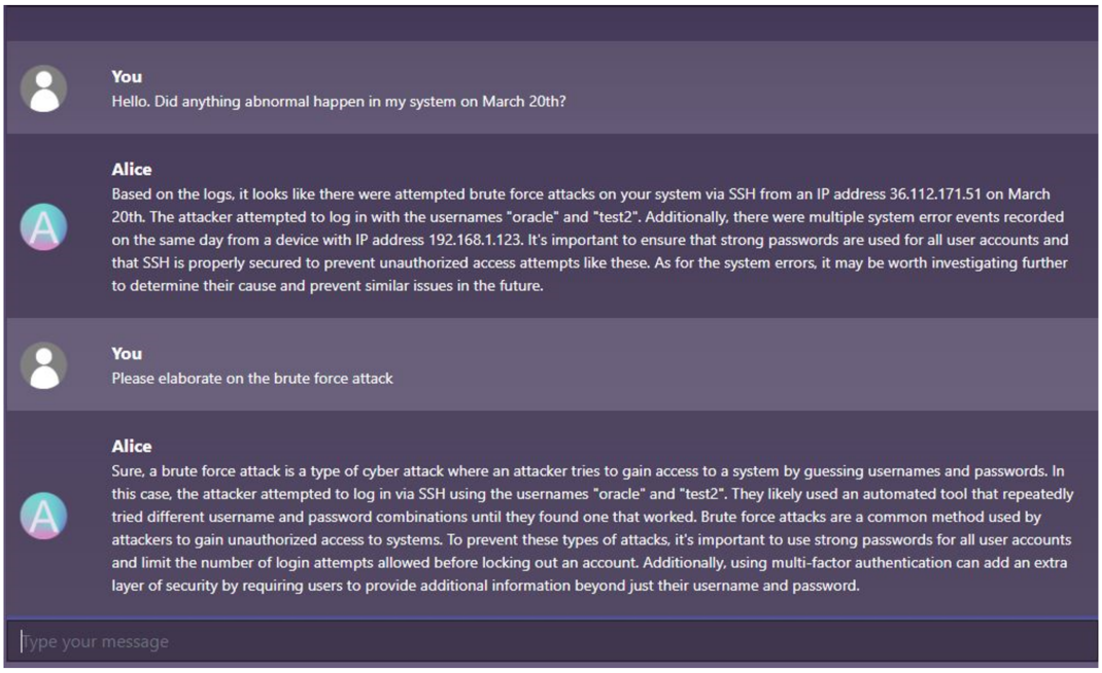
When initially scanning the user's input message, if any hardcoded keywords associated with blocking IP address response is detected in the user's input, the user input will then again get scanned for any IP addresses – if found, an API request to execute the block IP command on all agents is sent to the Wazuh manager. The main part of the response to that API request is analyzed and added to the end of the user's input with instructions for the AI. For example, if the user's input was "Block IP address 152.32.128.128" and the API request successfully blocked the IP address, the message getting sent to GPT-3.5 would be "*Block IP address 152.32.128.128 ***THIS IS WHAT HAPPENED WHEN YOU ATTEMPTED TO BLOCK THE IP ADDRESS: Block IP command was sent to all agents****". The AI will then generate a response to this message and notify the user what happened when the chatbot tried to block the IP address. See Figure 13 for an example interaction.


You
Hello. Could you block IP address 152.32.128.128

Alice
I have successfully blocked the IP address 152.32.128.128 on Wazuh agent 003. However, it seems that the command was not sent to agent 001 as it is not active. Is there anything else I can help you with?

Figure 13. Interaction between user and the chatbot, where the user wants the chatbot to block an IP address.

If the user's input contains any date in formats dd.mm.yyyy, dd.mm, yyyy.mm.dd, Jan 10, or 10th of January, the analyzing logs part of the program is activated. An API request for all system logs for the given date is sent to Elasticsearch. All system logs generated by Wazuh agents can be fetched via said Elasticsearch API. The logs get analyzed by our algorithm based on their threat level. All system logs generated by Wazuh have a threat level from 0 to 16. The higher the level, the more critical the log event is. The exact descriptions for each level can be found from Wazuh's documentation [22]. All logs at or above a specified threshold level (we used level 10 as the threshold, as this would eliminate nearly all meaningless logs from being forwarded to the AI) will have relevant information pulled from them, compacted, and included to end of the user's input before forwarding it to the AI. For example, if the user's input message is "Did anything abnormal happen in my system on 20th of March?", having our Wazuh manager connected to the chatbot, message: "****LOG DATA: *LOG EVENT: Device information: IP: 192.168.1.17, Wazuh agent name chatbotvm2, ID: 002, Full log: Mar 20 19:21:52 chatbotvm2 sshd[7654]: Disconnected from invalid user oracle 36.112.171.51 port 44652 [preauth],Mitre attack technique:['Brute Force'], mitre tactic: ['Credential Access'], Level: 10* *LOG EVENT: Device information: IP: 192.168.1.17, Wazuh agent name chatbotvm2, ID: 002, Full log: Mar 20 19:23:02 chatbotvm2 sshd[7691]: Disconnected from invalid user test2 36.112.171.51 port 49329 [preauth],Mitre attack technique:['Brute Force'], mitre tactic: ['Credential Access'], Level: 10* *LOG EVENT: Device information: IP: 192.168.1.123, Wazuh agent name DESKTOP-GV0CDGM, ID: 001, Description: Multiple System error events, Timestamp: 2023-03-20T21:36:50.120+0000, Level: 10* Analyzed 420 logs. Found 3 records of meaningful log events.****" will get added to the end of the user's input. The AI will then generate a response to this message and summarize the data from these log events. See Figure 14 for an example interaction.



You
Hello. Did anything abnormal happen in my system on March 20th?

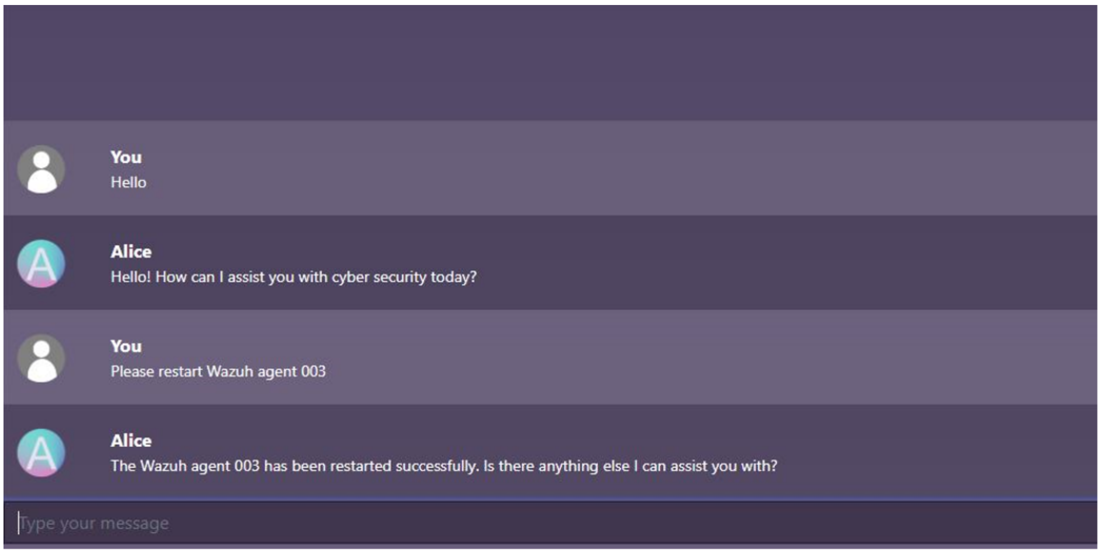
Alice
Based on the logs, it looks like there were attempted brute force attacks on your system via SSH from an IP address 36.112.171.51 on March 20th. The attacker attempted to log in with the usernames "oracle" and "test2". Additionally, there were multiple system error events recorded on the same day from a device with IP address 192.168.1.123. It's important to ensure that strong passwords are used for all user accounts and that SSH is properly secured to prevent unauthorized access attempts like these. As for the system errors, it may be worth investigating further to determine their cause and prevent similar issues in the future.

You
Please elaborate on the brute force attack

Alice
Sure, a brute force attack is a type of cyber attack where an attacker tries to gain access to a system by guessing usernames and passwords. In this case, the attacker attempted to log in via SSH using the usernames "oracle" and "test2". They likely used an automated tool that repeatedly tried different username and password combinations until they found one that worked. Brute force attacks are a common method used by attackers to gain unauthorized access to systems. To prevent these types of attacks, it's important to use strong passwords for all user accounts and limit the number of login attempts allowed before locking out an account. Additionally, using multi-factor authentication can add an extra layer of security by requiring users to provide additional information beyond just their username and password.

Figure 14. Interaction between user and chatbot, where the user wants to know if anything abnormal happened on their system on a specific date. The chatbot analyses all the logs generated by Wazuh for the given date and responds with a summary of notable events for that date.

If the user's input contains keywords associated with the restart Wazuh agents command, such as "restart", the user's input will get scanned again for any numeric values. IDs of Wazuh agents are numeric and in format "001". If a numeric value is also found in the user's input, a restart Wazuh agent command will be sent to the agent with corresponding ID via Wazuh's API. The response from this API request is analyzed and a summary of what happened when the API request was made is added to the end of the user's input. For example, if the user's input is "Restart agent 003", having our Wazuh manager connected to the chatbot, the message added to the user's input before forwarding it to the AI is "****THIS IS WHAT HAPPENED WHEN YOU ATTEMPTED TO RESTART THE WAZUH AGENT: Restart command was sent to agent 003****". The AI then generates a response to the user, letting the user know the restart command was sent to the agent successfully. See Figure 15 for an example interaction.



You
Hello

Alice
Hello! How can I assist you with cyber security today?

You
Please restart Wazuh agent 003

Alice
The Wazuh agent 003 has been restarted successfully. Is there anything else I can assist you with?

Figure 15. Interaction between user and the chatbot, where the user wants the chatbot to restart a Wazuh agent on their system.

With GPT-3.5 turbo, the maximum length of a conversation it can handle is 4096 tokens. This corresponds to roughly 3000 words and can be considered as a short-term memory of the AI. Every time the program receives a response from the AI, the token length of the conversation is also saved into a variable. Whenever the token length gets to over 3000 (starts nearing the maximum token length), the older half of the conversation between user and the chatbot is removed from memory, while also preserving the log data of latest analyzed logs if there are any. This allows the chatbot to continue conversation with the user to its maximum capability, while also still knowing the contents of latest logs it analyzed if the conversation between user and the chatbot gets to bigger lengths.

The backend uses Elasticsearch API to get Wazuh alerts for the given date. The Wazuh alerts are first sent from the Wazuh indexer to the backend using a curl request. The logs are then returned as a list.

Because the alerts contain some information that is not going to be useful for the bot, the backend will first extract the important details from the Wazuh alerts and create its own logs with only the useful information. The backend does that by scanning each alert one by one, and extracting the IP address, device name, Wazuh agent ID, timestamp, threat level, attack technique, and full log message or event description. The threat level is a number given by Wazuh that indicates how big the potential security risk is. The threat level is used for filtering the logs so that only the logs with higher threat level will be analyzed by the bot.

The blocking of IP addresses was implemented with the Wazuh API. Wazuh API makes it easier to block the same IP address on all devices that are being monitored. The backend will first create an authorization token that will allow the authorized usage of the Wazuh API. Afterwards the backend will be able to send a curl request with the authorization token that will block The IP address on all devices with Wazuh agent installed.

Restarting Wazuh agents works in a similar way. The only difference is that the user will also have to input the ID of the Wazuh agent they want to restart.

5. EVALUATION

5.1. Evaluation Methodology

In this chapter, we will discuss the methodology employed in the evaluation phase of our research, focusing on how the testing was conducted to assess the effectiveness and usability of our chatbot web application to assess system security. This evaluation aimed to gather feedback from a diverse group of users with varying degrees of experience in the fields of computers, cybersecurity, and artificial intelligence.

To conduct the testing, we recruited ten participants aged between 24 to 59 with different backgrounds from various communities. Our recruitment strategy included reaching out to people in Wazuh, cybersecurity, and ChatGPT-related subreddits on Reddit, as well as engaging with different communities where individuals can share and test new early-stage applications. This approach allowed us to gather a wide range of perspectives from both professionals and novices in the field.

Prior to the testing, we incorporated documentation and a Frequently Asked Questions (FAQ) section in our application to help users understand how to use the chatbot effectively. The testing period was initially set for three days, allowing participants to use and experiment with the application freely. However, due to technical difficulties and a possible Distributed Denial of Service (DDoS) attack on our application's servers, we extended the testing period to seven days to ensure all participants had ample time to explore and assess the chatbot's capabilities. Throughout the testing period, participants were encouraged to interact with the chatbot, asking questions, requesting analysis of their system logs, and performing tasks such as blocking IP addresses or restarting Wazuh agents, as needed. This hands-on approach provided valuable insights into the user experience and usability of the application.

Upon completion of the testing period, participants were asked to fill out a feedback survey containing questions regarding their experience with computers, cybersecurity, ChatGPT or other conversational AIs, and Wazuh, as well as their overall experience with the web application. They were also asked to provide insights into any bugs or issues encountered, the ease of use and responsiveness of the application, and suggestions for future improvements.

5.2. Analysis of Survey Data

The survey data from the feedback forms filled in by the participants was collected into a text file for analysis. We went through all the answers collected from our participants to find common, repeating themes amongst them. After our analysis of the answers, we found that there are three main categories and six subcategories.

The main categories are:

- Usability issues. These include issues where the participants ran into problems while using the application, such as the application not working at all or not working the way it was supposed to.

- Underdeveloped. This refers to participants feeling like the application was lacking functionalities or needed further development in certain areas.
- Usefulness. This refers to participants finding aspects of the application useful or helpful.

After identifying the themes, we calculated the results of how often each theme was present in the survey data according to our investigation of the material (Table 1). We then did agreement testing on the subcategories we had identified by having two people go through the survey data [23]. They were instructed to find the occurrences of the subcategories from the material given to them. We then calculated Cohen's kappa from the results we gathered from the agreement testing. We observed 46 cases, where both two people agreed on the same theme occurring on a specific sentence in the survey data. Since the six subcategories had in total occurred 50 times as shown in Table 1, the agreement percentage is 92%. The Cohen's kappa we calculated from the results was 0.721, which indicates substantial agreement. Most of the disagreements present in the agreement testing results did not come from any specific category, but rather from a small number of disagreements present in each category, except for the subcategories in the "Usability issues" category, in which both had identified the same occurrences.

Usability issues (n = 11)	Underdeveloped (n = 10)	Usefulness (n = 29)
Non-operational, application stops responding to user messages completely. (6)	Needs more functionalities and further development. (10)	Application is straightforward and easy to use. (9)
Application stops working the way it should by not performing tasks it is capable of, such as blocking IP address. (5)		Application is convenient for people who are inexperienced with cybersecurity. (11)
		Application gives good insight into cybersecurity for the user. (9)

Table 1. Results of how often each theme was present in the survey data.

5.3. Evaluation Results

In this section, we will go through the reasoning behind the inclusion of each question in the feedback survey and the goals we aimed to achieve by collecting feedback from participants through these questions. As well as analyze the feedback received from the subjects through the survey, highlighting key trends and insights that emerged from their responses.

By incorporating these questions into our survey, we aimed to gather comprehensive feedback that would enable us to assess the strengths and weaknesses of the chatbot, identify areas for development, and understand the potential value of the application for different user groups. This information is crucial for informing future design decisions and ensuring that the chatbot continues to evolve into an effective and user-friendly chatbot for assessing system security.

1.	How old are you?
2.	How would you rate your experience with computers on a scale of 1 to 10?
3.	How would you rate your understanding of cybersecurity on a scale of 1 to 10?
4.	How would you rate your experience with using ChatGPT, or other conversational AIs, on a scale of 1 to 10?
5.	How would you rate your experience with Wazuh on a scale of 1 to 10?
6.	Did you encounter any bugs or weird behavior during your testing?
7.	Please tell freely about your experience with the chatbot.
8.	Was the application easy to use and responsive?
9.	Are there any specific capabilities you would like the chatbot to have?
10.	Did you find the chatbot useful?
11.	Would you like to continue using the chatbot in the future?
12.	Do you think the chatbot would be a useful tool for cybersecurity professionals?
13.	Do you think the chatbot would be a useful tool for people with little experience with cybersecurity?

Table 2. Feedback questionnaire.

5.3.1. Questions 1-5: Background and Experience

The first four questions were designed to gather information about each participant's background and experience in relevant fields. Understanding their level of expertise in using computers, cybersecurity, conversational AIs (such as ChatGPT), and Wazuh enabled us to better interpret their feedback and identify any potential correlations between their experience and their perceptions of the chatbot. Moreover, these questions allowed us to ensure that our sample of testers represented a diverse range of users, which is crucial for a comprehensive evaluation.

The testers' responses to questions 1-4 revealed a diverse range of experience levels with computers, cybersecurity, conversational AIs, and Wazuh. This diversity allowed for a comprehensive evaluation of the chatbot, ensuring that the application was assessed from multiple perspectives.

5.3.2. Question 6: Bugs and Issues

In question five, we inquired about any bugs or weird behavior encountered during the testing. Identifying these issues is critical for refining and improving the application, ensuring that the chatbot functions smoothly and consistently for all users.

Testers reported some bugs and inconsistencies in the chatbot's behavior, such as forgetting its capabilities, and providing contradictory information. Addressing these issues is crucial for improving user experience and ensuring the chatbot's reliability.

5.3.3. Questions 7 and 8: Overall Experience and Usability

The sixth question invited participants to share their overall experience with the web application. This open-ended question enabled us to gain insights into their thoughts, feelings, and impressions regarding the chatbot, which helps us understand the factors that contribute to a positive or negative user experience.

Question seven aimed to assess the application's usability and responsiveness from the user's perspective. Understanding how easy and intuitive the application is to navigate and interact with can inform future design and development decisions, ultimately improving the overall user experience.

The majority of the testers found the chatbot to be useful and easy to use, particularly for non-professionals. The application was appreciated for its ability to provide valuable information about cybersecurity and for its responsiveness. However, some users experienced technical difficulties such as the chat feature not working on certain devices or the chatbot freezing. These issues highlight the need for further refinement and optimization of the application.

5.3.4. Question 9: Future Capabilities

The eighth question focused on gathering suggestions for specific features or capabilities that users would like to see in future iterations of the web application. This feedback is invaluable for guiding the direction of the chatbot's development, ensuring that it continues to evolve and meets the needs and expectations of users.

Participants provided suggestions for additional features, such as more detailed log analysis, advanced administrative actions, and integration with firewalls or antivirus software. These suggestions indicate that while the chatbot's current capabilities are helpful for some users, there is potential for further development and enhancement to cater to a wider audience.

5.3.5. Questions 10-13: Utility and Potential Use Cases

The final four questions sought to evaluate the perceived usefulness of the application, both for the participants themselves and for other potential user groups, such as cybersecurity professionals or individuals with limited cybersecurity knowledge. These questions aimed to gauge the overall value of the chatbot as a tool for various users, helping us understand its potential impact and areas for improvement to increase its utility and appeal to a wider audience.

Most testers found the chatbot to be useful, particularly for non-professionals or those with limited cybersecurity knowledge. However, some participants expressed concerns about the application's suitability for cybersecurity professionals, highlighting the need for additional features and improvements to cater to this user group. The feedback suggests that the chatbot has potential as a valuable tool for both professionals and novices, but further development is required to fully realize this potential.

5.3.6. Conclusion of Feedback Analysis

In conclusion, the analysis of the survey feedback reveals that the chatbot holds promise for assessing system security for a diverse range of users. However, there are areas for improvement, including addressing technical issues, enhancing the chatbot's capabilities, and ensuring the application caters to the needs of both professionals and novices in the field. By leveraging the insights gained from the feedback, we can continue to refine and develop the chatbot to better serve the needs of its users and increase its overall utility and effectiveness.

5.4. Limitations

The study did not include a comprehensive qualitative analysis. The findings are based solely on survey data analysis and agreement testing, which are preliminary in nature and may not capture the richness and depth of qualitative insights that could have been obtained through a more comprehensive analysis.

The small size of the sample used in this study may restrict the generalizability of the findings to a broader population. A larger participant group would have provided more diverse perspectives and potentially yielded more insightful results.

The participants were primarily recruited from a single social media platform, which may introduce inherent biases. Including participants from a wider range of platforms would have diversified the data pool and reduced the likelihood of skewed results influenced by the demographic characteristics of a single platform.

6. CONCLUSION

In this thesis, we have explored the potential of using conversational AI, specifically ChatGPT based on GPT-3.5, for assessing system security. Our focus was on developing a chatbot that could assist users in navigating the complex landscape of cybersecurity by analyzing log data from a host-based intrusion detection system and informing users of any potential threats or anomalies detected.

Our implementation of a web application chatbot that integrates with Wazuh to collect logs, block IP addresses and restart agents on user's devices when necessary, showed promising results. The chatbot was able to provide valuable insights and answers to users' questions regarding their system security, which is critical in today's digital age where cybersecurity threats are constantly evolving.

The evaluation of our chatbot through survey feedback revealed that it holds promise for assessing system security for a diverse range of users. However, there are areas for improvement and future development, such as enhancing the chatbot's capabilities, and ensuring the application caters to the needs of both professionals and novices in the field.

The use of conversational AI for assessing system security has great potential and the development of our chatbot demonstrates its feasibility. As the field of AI continues to evolve, it is important to explore the different ways this technology can be leveraged to improve cybersecurity and make our digital lives more secure.

7. REFERENCES

[1] Similarweb statistics (2023). URL: https://www.similarweb.com/website/chat.openai.com/#overview. Accessed 16.2.2023.

[2] OpenAI (2023), Documentation. URL: https://platform.openai.com/docs/models/gpt-3-5. Accessed 11.4.2023.

[3] Destefanis G., Bartolucci S. & Ortu M. (2023), A Preliminary Analysis on the Code Generation Capabilities of GPT-3.5 and Bard AI Models for Java Functions. URL: https://arxiv.org/abs/2305.09402. Accessed 22.6.2023.

[4] Collins E. & Ghahramani Z. (2021), LaMDA: our breakthrough conversation technology. Google Blog. URL: https://blog.google/technology/ai/lamda/. Accessed 16.2.2023.

[5] Pichai S. (2023), An important next step on our AI journey. Google Blog. URL: https://blog.google/technology/ai/bard-google-ai-search-updates/. Accessed 16.2.2023.

[6] Goodside R. & Papay S. (2023), Meet Claude: Anthropic's Rival to ChatGPT. Scale Blog. URL: https://scale.com/blog/chatgpt-vs-claude. Accessed 15.2.2023.

[7] Glaese A., McAleese N., Trebacz M., Aslanides J., Firoiu V., Ewalds T., Rauh M., Weidinger L., Chadwick M., Thacker P., Campbell-Gillingham L., Uesato J., Huang P.S., Comanescu R., Yang F., See A., Dathathri S., Greig R., Chen C., Fritz D., Elias J., Green R., Mokrá S., Fernando N., Wu B., Foley R., Young S., Gabriel I., Isaac W., Mellor J., Hassabis D., Kavukcuoglu K., Hendricks L. & Irving G. (2022), Improving alignment of dialogue agents via targeted human judgements. URL: https://arxiv.org/pdf/2209.14375.pdf. Accessed 15.2.2023.

[8] Bai Y., Kadavath S., Kundu S., Askell A., Kernion J., Jones A., Chen A., Goldie A., Mirhoseini A., McKinnon C., Chen C., Olsson C., Olah C., Hernandez D., Drain D., Ganguli D., Li D., Tran-Johnson E., Perez E., Kerr J., Mueller J., Ladish J., Landau J., Ndousse K., Lukosuite K., Lovitt L., Sellitto M., Elhage N., Schiefer N., Mercado N., DasSarma N., Lasenby R., Larson R., Ringer S., Johnston S., Kravec S., El Showk S., Fort S., Lanham T., Telleen-Lawton T., Conerly T., Henighan T., Hume T., Bowman S., Hatfield-Dodds Z., Mann B., Amodei D., Joseph N., McCandlish S., Brown T. & Kaplan J. (2022), Constitutional AI: Harmlessness from AI Feedback. URL: https://arxiv.org/pdf/2212.08073.pdf. Accessed 15.2.2023.

[9] The Sparrow Team (2022), Building safer dialogue agents. DeepMind Blog. URL: https://www.deepmind.com/blog/building-safer-dialogue-agents. Accessed 15.2.2023.

[10] Wazuh Inc. (2015-2023), Documentation: Elasticsearch. URL: https://documentation.wazuh.com/current/user-manual/elasticsearch/index.html. Accessed 22.6.2023.

[11] Wazuh Inc. (2015-2023), Documentation: Kibana. URL: https://documentation.wazuh.com/current/deployment-options/elastic-stack/distributed-deployment/kibana/index.html. Accessed 22.6.2023.

[12] Wazuh Inc. (2015-2023), Documentation. URL: https://documentation.wazuh.com/current/index.html. Accessed 16.2.2023.

[13] Wazuh Inc. (2015-2023), API Documentation. URL: https://documentation.wazuh.com/current/user-manual/api/reference.html. Accessed 19.2.2023.

[14] OpenAI (2023), Documentation on Fine-Tuning. URL: https://platform.openai.com/docs/guides/fine-tuning. Accessed 20.2.2023.

[15] Drawing Vector Robot PNG Transparent Background. URL: https://www.freeiconspng.com/img/30475. Accessed 22.2.2023.

[16] CSC Services: cPouta. URL: https://research.csc.fi/-/cpouta. Accessed 22.6.2023.

[17] Boduch A. (2017) React and React Native. Packt Publishing Ltd.

[18] You, E. & Vite Contributors (2019-2023), Vite: Next Generation Frontend. URL: https://vitejs.dev/. Accessed 22.6.2023.

[19] Vite (2019-2023), Documentation. URL: https://vitejs.dev/guide/. Accessed 27.3.2023.

[20] Facebook, Inc (2022), Create React App. URL: https://create-react-app.dev/. Accessed 22.6.2023.

[21] Gawai C. (2022), 4 Reasons Why You Should Prefer Vite Over Create-React-App (CRA). Semaphore Blog. URL: https://semaphoreci.com/blog/vite. Accessed 27.3.2023.

[22] Wazuh Inc. (2015-2023), Documentation: Rules classification. URL: https://documentation.wazuh.com/current/user-manual/ruleset/rules-classification.html. Accessed 27.3.2023.

[23] Bakeman R. & Quera V. (2011) Sequential Analysis and Observational Methods for the Behavioral Sciences. Cambridge University Press.